Symbolische BDD-basierte Modellprüfung asynchroner nebenläufiger Systeme

Dissertation zur Erlangung des
naturwissenschaftlichen Doktorgrades
der Bayerischen Julius-Maximilians-Universität Würzburg

vorgelegt von

Christian Appold

aus
Stuppach

Würzburg 2015

Eingereicht am: 14.10.2015
bei der Fakultät für Mathematik und Informatik
1. Gutachter: Prof. Dr. Reiner Kolla
2. Gutachter: Prof. Dr. Christoph Scholl

Danksagung

Die der vorliegenden Arbeit zugrunde liegenden Forschungsarbeiten wurden am Lehrstuhl für Informatik V der Julius-Maximilians-Universität Würzburg durchgeführt. Für die Möglichkeit an seinem Lehrstuhl promovieren und arbeiten zu dürfen, möchte ich meinem Betreuer Prof. Dr. Reiner Kolla danken. Ebenso möchte ich Prof. Dr. Christoph Scholl für die Durchführung der Zweitkorrektur und die dafür aufgebrachte Zeit danken.

Dank gilt auch meinen Korrekturlesern Hansjörg Herrmann, Armin Runge, Benedikt Appold und Dr. Jürgen Bregenzer. Durch Ihre nützlichen Korrekturen und Verbesserungsvorschläge haben Sie einen wertvollen Beitrag zum Gelingen dieser Arbeit geleistet. Für das gute Arbeitsklima am Lehrstuhl und für viele anregende Diskussionen, möchte ich meinen ehemaligen Arbeitskollegen Prof. Dr. Marcel Baunach, Clemens Mühlberger, Dr. Jürgen Bregenzer und Armin Runge danken.

Besonderer Dank gilt auch meinen Eltern für Ihre Unterstützung. Außerdem danke ich meinen Freunden, die mir in schweren Zeiten ehrlich beigestanden haben und mit denen ich einige tolle Stunden verbringen durfte. Hier möchte ich Frank Feustel hervorheben.

Abstract

Today, information and communication systems are ubiquitous and consist very often of several interacting and communicating components. One reason is the widespread use of multi-core processors and the increasing amount of concurrent software for the efficient usage of multi-core processors. Also, the dissemination of distributed emergent technologies like sensor networks or the internet of things is growing. Additionally, a lot of internet protocols are client-server architectures with clients which execute computations in parallel and servers that can handle requests of several clients in parallel. Systems which consist of several interacting and communicating components are often very complex and due to their complexity also prone to errors. Errors in systems can have dramatic consequenses, especially in safety-critical areas where human life can be endangered by incorrect system behavior. Hence, it is inevitable to have methods that ensure the proper functioning of such systems.

This thesis aims on improving the verifiability of asynchronous concurrent systems using symbolic model checking based on Binary Decision Diagrams (BDDs). An asynchronous concurrent system is a system that consists of several components, from which only one component can execute a transition at a time. Model checking is a formal verification technique. For a given system description and a set of desired properties, the validity of the properties for the system is decided in model checking automatically by software tools called model checkers. The main problem of model checking is the state-space explosion problem. One approach to reduce this problem is the use of symbolic model checking. There, system states and transitions are not stored explicitely as in explicit model checking. Instead, in symbolic model checking sets of states and sets of transitions are stored and also manipulated together. The data structure which is used in this thesis to store those sets are BDDs. BDD-based symbolic model checking has already been used successful in industry for several times. Nevertheless, BDD-based symbolic model checking still suffers from the state-space explosion problem and further improvements are necessary to improve its applicability.

Central operations in BDD-based symbolic model checking are the computation of successor and predecessor states of a given set of states. Those computations are called image computations. They are applied repeatedly in BDD-based symbolic model checking to decide the validity of properties for a given system description. Hence, their efficient execution is crucial for the memory and runtime requirements of a model checker. In an image computation a BDD for a set of transitions and a BDD for a set of states are combined to compute a set of successor or predecessor states. Often, also the size of the BDDs to represent the transition relation ist critical for the successful use of model checking. To further improve the applicability of symbolic model checking, we present in this thesis new data structures to store the transition relation of asynchronous concurrent systems. Additionally, we present new image computation algorithms. Both can lead to large runtime and memory reductions for BDD-based symbolic model checking. Asynchronous concurrent systems often contain symmetries. A technique to exploit those symmetries to diminish the state-space explosion problem is symmetry reduction. In this thesis we also present a new efficient algorithm for symmetry reduction in BDD-based symbolic model checking.

Inhaltsverzeichnis

1 Einleitung

1.1 Problemstellung

In unserem Alltag kommen wir heute ständig mit Systemen der Informations- und Kommunikationstechnik (IKT) in Kontakt. Ein Beispiel für solche Systeme sind Eingebettete Systeme. Diese begegnen uns unter anderem in der Automobiltechnik, der Luft- und Raumfahrttechnik, der Telekommunikationstechnik, der Medizintechnik und in Haushaltsgeräten. Es sind aber beispielsweise auch Computer, sowie Software zur Kommunikation im Internet oder Software zur medizinischen Bildverarbeitung Systeme der Informations- und Kommunikationstechnik.

Neben ihrer weiten Verbreitung steigt die Komplexität von Systemen der Informations- und Kommunikationstechnik immer mehr an. Oft bestehen solche Systeme aus mehreren Komponenten, die miteinander kommunizieren und interagieren, oder sie arbeiten mit möglicherweise mehreren anderen Komponenten und Systemen zusammen. Mit der zunehmenden Komplexität von Systemen steigt auch die Wahrscheinlichkeit für in ihnen vorhandene Fehler. Diese Fehler können zu großen finanziellen Schäden oder sogar zur Gefährdung von Menschenleben führen. Einige bekannte Fehler und deren Folgen werden im Folgenden aufgeführt. Ein Beispiel ist der Mitte der neunziger Jahre des letzten Jahrhunderts entdeckte Fehler in der Gleitkommadivisionseinheit des Intel Pentium. Dieser verursachte Intel einen Schaden von 475 Millionen US-Dollar für die Ersetzung von fehlerhaften Prozessoren. Zusätzlich wurde der Ruf der Firma als zuverlässiger Chipfabrikant stark beschädigt [8]. Im Jahr 1996 führte ein Fehler in der Kontrollsoftware der Ariane-5 Rakete zur Explosion dieser kurz nach ihrem Start. Neben einem großen finanziellen Schaden resultierte daraus eine Zeitverzögerung, da es einige Zeit in Anspruch nahm bis der dadurch notwendige erneute Start durchgeführt werden konnte. Zwischen 1985 und 1987 führte beim Bestrahlungsgerät Therac-25 ein Softwarefehler, durch den Patienten einer zu hohen Strahlendosis ausgesetzt wurden, zum Tod von sechs Krebspatienten [8].

Fehler in Systemen der Informations- und Kommunikationstechnik können somit zu katastrophalen Folgen führen. Aufgrund der weiten Verbreitung und der steigenden Komplexität solcher Systeme, ist es unerlässlich Methoden zu besitzen und zu entwickeln, die helfen deren Zuverlässigkeit zu steigern. Ganz besonders trifft dies für Systeme in sicherheitskritischen Bereichen zu, in denen Menschenleben direkt oder indirekt vom Funktionieren dieser Systeme abhängen. Möglichkeiten den Entwurf zuverlässig funktionierender Systeme zu unterstützen sind die Verwendung von Entwurfstechniken, die darauf abzielen das Entstehen von Fehlern von vorneherein zu vermeiden, oder das Benutzen von Techniken zur Erkennung und Beseitigung von vorhandenen Fehlern. In der vorliegenden Arbeit werden Verbesserungen zur Modellprüfung [53, 163] (siehe auch Abschnitt 2.1) asynchro-

ner nebenläufiger Systeme präsentiert. Die Modellprüfung ist ein Ansatz zur Verifikation von Systemen. Es wird bei ihr untersucht, ob bestimmte vorher angegebene Eigenschaften für alle Verhaltensweisen eines Systemmodells gültig sind. Ist dies nicht der Fall und eine nicht erfüllte Eigenschaft ist auf einen Fehler im verwendeten Systemmodell zurückzuführen, wurde ein Fehler im Systemmodell und möglicherweise auch im diesem zugrunde liegenden System gefunden. Da es sich bei der in der vorliegenden Arbeit behandelten Modellprüfung um ein Verfahren mit dem Fehler entdeckt werden können handelt, werden Entwurfstechniken im Folgenden nicht weiter behandelt. In den hier betrachteten asynchronen nebenläufigen Systemen kann von mehreren vorhandenen Komponenten zu einem Zeitpunkt nur eine Komponente eines Systems Transitionen ausführen. Das aufeinanderfolgende und verschachtelte Ausführen von Transitionen durch Komponenten führt meistens zu einer großen Anzahl an möglichen verschiedenen Verschachtelungen von Ausführungen von Transitionen. Von diesen führen häufig aber nur wenige Verschachtelungen zu den typischen Fehlertypen bei vorhandener Nebenläufigkeit, was die Entdeckung dieser Fehler in solchen Systemen erschwert. Einige typische Fehler bei vorhandener Nebenläufigkeit werden später in diesem Abschnitt beschrieben.

Es gibt verschiedene Verifikationsverfahren, die zur Steigerung der Zuverlässigkeit von Systemen eingesetzt werden können. Allgemein prüfen Verifikationsverfahren ein System auf die Gültigkeit vorher angegebener Eigenschaften. Diese können zum Beispiel aus der Spezifikation eines Systems abgeleitet worden sein. Wird entdeckt, dass ein System eine gewünschte Eigenschaft nicht erfüllt, wurde ein Fehler im System gefunden. Häufig eingesetzte Verifikationsverfahren sind zum Beispiel das Peer-Reviewing, die Simulation, das Testen, die deduktive Verifikation und die bereits erwähnte Modellprüfung. Beim Peer-Reviewing werden Designdokumente und Quellcode (z.B. von Software oder von einem in einer Hardwarebeschreibungssprache beschriebenen Hardwaredesign) zu einem System von einer Gruppe von Entwicklern, die im Idealfall nicht bei der Entwicklung des Systems beteiligt sind, begutachtet. Ein Ziel der Anwendung von Peer-Reviewing ist es, Fehler so früh wie möglich zu entdecken, um durch deren Behebung entstehende Kosten möglichst gering zu halten. Durch Studien wurde belegt, dass Peer-Reviewing eine kostengünstige und effiziente Technik zum Auffinden von Fehlern ist. Da es sich dabei allerdings um eine statische Technik handelt bei der keine Ausführung von Quellcode stattfindet, lassen sich damit subtile Fehler, wie typische Fehler bei vorhandener Nebenläufigkeit (siehe Beschreibung solcher Fehler weiter unten) oder Fehler in Algorithmen, schwer finden [8]. Bei der Simulation und dem Testen handelt es sich um dynamische Techniken, bei denen Ausführungen von Systemen betrachtet werden. Dabei unterscheiden sich die Simulation und das Testen hauptsächlich dadurch, dass bei der Simulation mit einem Modell oder einer Abstraktion eines Systems gearbeitet wird, wohingegen das Testen mit dem tatsächlichen Produkt durchgeführt wird. Es wird bei beiden eine Stimulation von Systemen mit typischen Verarbeitungsabläufen entsprechenden Eingaben durchgeführt und beobachtet, ob daraus resultierende Verhaltensweisen von Systemen den gewünschten Verhaltensweisen entsprechen. Um dadurch immer zutreffende Aussagen über ein untersuchtes System treffen zu können, müssten alle möglichen Verhaltensweisen des Systems untersucht werden. Dies könnte durch eine vollständige Simulation bzw. ein vollständiges Testen durchge-

führt werden, bei denen Systeme mit allen möglichen Eingabefolgen stimuliert werden. Allerdings ist dies praktisch kaum möglich. In nebenläufigen Systemen mit mehreren interagierenden und kommunizierenden Komponenten treten bestimmte Fehler oftmals nur bei ganz speziellen Ausführungsreihenfolgen der Komponenten auf. Da bei der Simulation und dem Testen gewöhnlich nur eine Teilmenge aller möglichen Verhaltensweisen von Systemen untersucht werden kann, können solche Fehler durch diese ebenfalls nur schwer gefunden werden.

Die deduktive Verifikation und die Modellprüfung sind Techniken zur formalen Verifikation. Bei der formalen Verifikation wird die korrekte Funktionsweise von Systemen mit mathematisch exakten Methoden gezeigt, wodurch zusätzlich zum Auffinden von Fehlern auch die Fehlerfreiheit gezeigt werden kann. Untersuchungen haben erwiesen, dass die beim Intel Pentium-Prozessor, der Ariane-5 Rakete oder der Therac-25 Bestrahlungsmaschine aufgetretenen Fehler bei Anwendung von Methoden zur formalen Verifikation entdeckt worden wären [8]. Verfahren zur deduktiven Verifikation (siehe z.B. [89, 114, 127]) benutzen Axiome und Schlussregeln, um die Korrektheit von Systemen zu beweisen. In den ursprünglichen Ansätzen dazu wurden die Beweise manuell durchgeführt. Mittlerweile existieren aber Softwarewerkzeuge, die das halbautomatische und automatische Erstellen von Beweisen ermöglichen. Der Nachteil dieser Verfahren ist, dass sie meistens nur von Experten mit dem dazu notwendigen Fachwissen angewandt werden können. Außerdem kann das Erstellen von Beweisen sehr zeitaufwendig sein. Bei der Modellprüfung wird nach Eingabe eines Systemmodels und einer Menge von gewünschten Eigenschaften durch einen Benutzer geprüft, ob diese Eigenschaften für alle möglichen Verhaltensweisen des Systemmodells gültig sind. Das Überprüfen der Eigenschaften wird dabei vollautomatisch von Softwarewerkzeugen, sogenannten Modellprüfern, durchgeführt. Dadurch ist die Modellprüfung einfacher anzuwenden als Verfahren der deduktiven Verifikation. Dies ist ein Grund, warum die Modellprüfung in der Industrie immer häufiger bei der Verifikation von Systemen verwendet wird. Die Gültigkeit von Eigenschaften für Systeme wird von Modellprüfern durch spezielle Algorithmen nachgewiesen, die systematisch alle erreichbaren Systemzustände eines eingegebenen Systems explorieren. Das Hauptproblem der Modellprüfung ist das Problem der Zustandsraumexplosion, da der zu untersuchende Zustandsraum exponentiell mit der Anzahl der im System vorhandenen Variablen und in nebenläufigen Systemen auch exponentiell mit der Anzahl im System vorhandenen Komponenten wächst [8]. Neben der einfachen Anwendbarkeit ist ein weiterer Vorteil der Modellprüfung, dass Modellprüfer bei nicht erfüllten Eigenschaften entsprechende Ausführungspfade des Systems als Gegenbeispiele ausgeben. Diese können Systementwicklern nützliche Informationen zur Behebung von Fehlern in Systemen liefern. Mehr Informationen zur Modellprüfung sind im weiteren Verlauf dieser Arbeit zu finden.

In der vorliegenden Arbeit werden Verbesserungen zur auf binären Entscheidungsdiagrammen (BDDs, siehe Abschnitt 2.2) basierenden symbolischen Modellprüfung asynchroner nebenläufiger Systeme vorgestellt. Die ersten entwickelten Verfahren zur Modellprüfung waren Verfahren zur expliziten Modellprüfung, bei denen Zustände und Transitionen von Systemen explizit abgespeichert wurden. Bei der symbolischen Modellprüfung wird hingegen mit Mengen von Zuständen und mit Mengen von Transitionen gearbeitet. Ei-

ne ausführlichere Beschreibung der expliziten und der symbolischen Modellprüfung ist in Abschnitt 2.1.4 zu finden. BDDs sind eine Datenstruktur, die bei der BDD-basierten symbolischen Modellprüfung zur Repräsentation von Mengen von Zuständen und von Mengen von Transitionen verwendet wird. Häufig können diese mit BDDs kompakt repräsentiert werden. Zusätzlich existieren für BDDs effiziente Algorithmen für die zur BDD-basierten Modellprüfung notwendigen Mengenoperationen. Asynchrone nebenläufige Systeme sind heute bereits weit verbreitet und ihre Verbreitung nimmt immer mehr zu. Ein Grund dafür ist die zunehmende Verwendung von Mehrkern-Prozessoren. Um für Softwareimplementierungen möglichst große Effizienzsteigerungen durch die Nutzung von Mehrkern-Prozessoren erzielen zu können, wird immer mehr nebenläufige Software entwickelt. Solche Software besteht aus mehreren parallel ausgeführten Abarbeitungsfäden. Diese können gleichzeitig auf verschiedenen Kernen oder stückweise auf einem Kern ausgeführt werden. Außerdem sind viele Systeme heute an sich nebenläufige Systeme, die aus mehreren miteinander interagierenden und kommunizierenden Komponenten bestehen. Ein Beispiel hierfür sind Sensornetzwerke, die viele Sensorknoten umfassen können und die zusammenarbeiten, um gemeinsam bestimmte Aufgaben zu erfüllen. Ebenso handelt es sich bei Client-Server Architekturen, bei denen mehrere Clients parallel Berechnungen durchführen und Server Anfragen von Clients parallel abarbeiten können, um nebenläufige Systeme. Auf Client-Server Architekturen basieren zum Beispiel viele Internet-Protokolle. Eine formale Definition der in dieser Arbeit betrachteten asynchronen nebenläufigen Systeme ist in Abschnitt 2.3 zu finden.

In asynchronen nebenläufigen Systemen häufig vorkommende Fehler sind Fehler bei der Synchronisation und Kommunikation von Komponenten. Ein Beispiel dafür ist, wenn zwei oder mehrere Komponenten eines Systems gemeinsamen Lese- und Schreibzugriff auf Daten haben und das Endergebnis von der Ausführungsreihenfolge der Transitionen der Komponenten abhängig ist. In diesem Fall liegt ein Fehler im System vor, wenn das System Ausführungsreihenfolgen von Transitionen der Komponenten zulässt, die zu nicht gewünschten und damit falschen Ergebnissen von Berechnungen führen. Ein weiterer typischer bei asynchronen nebenläufigen Systemen auftretender Fehlertyp sind Deadlocks. Diese führen zum Blockieren des Systems oder von Teilen des Systems und können Systeme funktionsunfähig machen. Fehler in asynchronen nebenläufigen Systemen können schlimme Auswirkungen haben und typische durch Nebenläufigkeit verursachte Fehler treten oft nur bei wenigen Verschachtelungen von Ausführungen von Transitionen von Komponenten auf. Daher sind solche Fehler nur schwer zu entdecken und es ist wichtig zuverlässige Verfahren zur Verifikation solcher Systeme zu besitzen, um deren Funktionieren sicherzustellen. Da bei der Simulation oder dem Testen nicht alle Verhaltensweisen von Systemen untersucht werden, sind nur bei wenigen Verschachtelungen auftretende Fehler mit diesen meistens nur schwer zu finden. Das gleiche Problem trifft auch auf das Peer-Reviewing zu, da dort keine Ausführung von Quellcode stattfindet. Deduktive Verfahren sind aufgrund ihres Zeitaufwands und des nötigen Expertenwissens im industriellen Umfeld häufig nicht geeignet. Dahingegen ist die Modellprüfung ein vollautomatisches Verfahren, bei dem alle Verhaltensweisen von Systemen untersucht werden. Die Modellprüfung ist deshalb zur Verifikation von asynchronen nebenläufigen Systemen gut geeignet. Allerdings leidet die

Modellprüfung unter dem Problem der Zustandsraumexplosion und in asynchronen nebenläufigen Systemen wächst die Anzahl der möglichen Systemzustände nicht nur exponentiell mit der Anzahl der im System vorhandenen Variablen, sondern auch exponentiell mit der Anzahl der im System vorhandenen Komponenten [8]. Daher ist die Entwicklung von Verfahren zur Abmilderung des Problems der Zustandsraumexplosion bei der Modellprüfung von asynchronen nebenläufigen Systemen sehr wichtig. Dazu wurden bereits einige Techniken entwickelt, wie zum Beispiel die Verwendung der symbolischen Modellprüfung anstatt der expliziten Modellprüfung oder Techniken zu Ausnutzung von in Systemen vorhandenen Symmetrien (siehe Abschnitt 2.4). Trotz dieser Verbesserungen sind Speicherbedarf und Laufzeit der Modellprüfung oft noch sehr hoch, was das erfolgreiche Anwenden der Modellprüfung verhindern kann. Das Ziel dieser Arbeit war daher die Entwicklung von Verfahren, die die Verifizierbarkeit von asynchronen nebenläufigen Systemen durch die BDD-basierte symbolische Modellprüfung weiter verbessern. Für die BDD-basierte symbolische Modellprüfung werden in der vorliegenden Arbeit neue Datenstrukturen und Algorithmen vorgestellt. Zusätzlich wurde ein effizienterer Algorithmus zur Ausnutzung von in Systemen vorhandenen Symmetrien entwickelt. Der Aufbau dieser Arbeit und die im Rahmen der Arbeit entwickelten neuen Ansätze werden im nachfolgenden Abschnitt beschrieben.

1.2 Aufbau der Arbeit und Beitrag

In der vorliegenden Arbeit werden in Kapitel 2 zuerst einschlägige Vorarbeiten zu den behandelten Themenbereichen aufgeführt. Dazu wird in Abschnitt 2.1 die Modellprüfung betrachtet. Es werden dort ein formales Modell eines endlichen Systems sowie temporale Logiken, die bei der Modellprüfung häufig zur Beschreibung von für Systeme zu prüfenden Eigenschaften verwendet werden, vorgestellt. Außerdem werden die explizite und die symbolische Modellprüfung und Algorithmen für die symbolische Modellprüfung beschrieben. Das Ziel der vorliegenden Arbeit war die Entwicklung von Verbesserungen zur auf binären Entscheidungsdiagrammen (engl. Binary Decision Diagrams, BDDs) basierenden symbolischen Modellprüfung asynchroner nebenläufiger Systeme. BDDs sind eine Datenstruktur, die bei der auf BDDs basierenden symbolischen Modellprüfung zur Repräsentation von Mengen von Zuständen und von Mengen von Transitionen von Systemen verwendet wird. Bei der BDD-basierten symbolischen Modellprüfung wird eine kanonische Variante von BDDs, sogenannte reduzierte geordnete binäre Entscheidungsdiagramme (ROBDDs), verwendet. ROBDDs, verfügbare Programmpakete in denen ROBDDs und Operationen mit diesen implementiert wurden und für die Modellprüfung mit ROBDDs wichtige Algorithmen zu deren Kombination werden in Abschnitt 2.2 vorgestellt. Anschließend wird in Abschnitt 2.3 der in dieser Arbeit betrachtete Systemtyp der endlichen asynchronen nebenläufigen Systeme definiert. Diese bestehen aus mehreren Komponenten, die unabhängig voneinander Transitionen ausführen können, wobei zu einem Zeitpunkt nur eine Komponente Transitionen ausführen darf.

Die Ausnutzung von Symmetrien bei der Modellprüfung wird in Abschnitt 2.4 beschrieben. Bei Anwendung der Modellprüfung für asynchrone nebenläufige Systeme tritt

das Problem der Zustandsraumexplosion besonders stark auf, da die Anzahl der Systemzustände hier nicht nur exponentiell mit der Anzahl der Variablen, sondern auch exponentiell mit der Anzahl der im System vorhandenen Komponenten ansteigt. In asynchronen nebenläufigen Systemen und besonders in asynchronen nebenläufigen Systemen mit replizierten Komponenten sind aber oft viele Symmetrien vorhanden. Verfahren zur Symmetriereduktion versuchen diese Symmetrien auszunutzen, indem sie die Durchmusterung des Zustandsraums auf Repräsentanten von Äquivalenzklassen von Zuständen mit unter den vorhandenen Symmetrien gleichen Eigenschaften beschränken. Dadurch kann das Problem der Zustandsraumexplosion abgemildert werden. Zusätzlich ergeben sich oft große Laufzeitverbesserungen. Zur Symmetriereduktion werden zuerst Grundlagen angegeben, bevor auf Verfahren zur Symmetriereduktion für die explizite und die symbolische Modellprüfung eingegangen wird. Anschließend werden Modellprüfer, für die die Ausnutzung von Symmetrien implementiert wurde, vorgestellt. Bei der BDD-basierten Modellprüfung werden zur Exploration aller Zustände eines Systems wiederholt Bildberechnungen mit Mengen von Zuständen und mit Mengen von Transitionen ausgeführt, bis alle erreichbaren Zustände ermittelt wurden. Um dies möglichst effizient durchführen zu können wurden bereits einige Ansätze entwickelt. Auf die Durchführung von Bildberechnungen bei der BDD-basierten Modellprüfung und auf diese Ansätze wird in Abschnitt 2.5 eingegangen.

Beginnend mit Kapitel 3 werden die im Rahmen der vorliegenden Arbeit neu entwickelten Verbesserungen zur BDD-basierten Modellprüfung asynchroner nebenläufiger Systeme beschrieben. Um diese verständlicher zu präsentieren werden in den entsprechenden Kapiteln Beispiele aufgeführt, mit denen entwickelte Datenstrukturen und Algorithmen veranschaulicht werden. Diese Beispiele werden zu einem Beispielsystem angegeben, welches in Abschnitt 2.6 vorgestellt wird. Bei der Modellprüfung werden zur Exploration des Zustandsraums wiederholt Bildberechnungen durchgeführt, bei denen Vorgängerzustände bzw. Nachfolgerzustände von Zuständen ermittelt werden. Eine Bildberechnung besteht bei der BDD-basierten Modellprüfung aus mehreren Operationen mit BDDs, die im naiven Ansatz hintereinander ausgeführt werden. Ein Algorithmus, der diese Operationen gemeinsam berechnet, wird in Abschnitt 3 präsentiert. Dessen Anwendung kann zu großen Verbesserungen der Laufzeit und des benötigten Speicherbedarfs führen.

In Kapitel 4 wird die neue Datenstruktur der Transitionslokalitätsausnutzenden binären Entscheidungsdiagramme (TLEBDDs) vorgestellt. Diese sind eine Datenstruktur zur platzsparenderen Repräsentation der Transitionen einer Komponente eines asynchronen nebenläufigen Systems mit Transitionslokalität. Transitionslokalität bedeutet dabei, dass Transitionen einer Komponente nur auf die globalen Variablen eines Systems und die eigenen lokalen Variablen der Komponente zugreifen dürfen (siehe auch Kapitel 2.3). Zusätzlich wird ein Algorithmus zur effizienten Bildberechnung mit TLEBDDs präsentiert. Eine ebenfalls neu entwickelte Datenstruktur, die eine Erweiterung der TLEBDDs ist und in denen Transitionen mehrerer Komponenten gemeinsam gespeichert werden können, sind erweiterte Transitionslokalitätsausnutzende binäre Entscheidungsdiagramme (ETLEBDDs). ETLEBDDs und ein Algorithmus zur effizienten Bildberechnung mit ETLEBDDs werden in Kapitel 5 beschrieben. Dieser Algorithmus erlaubt bei einem Aufruf die gleichzeitige Ausführung von Bildberechnungen für mehrere Komponenten, unter ein-

maliger Ausführung von für mehrere Komponenten gleich anfallenden Teilberechnungen. Zusätzlich werden in Kapitel 5 verschiedene Strategien zur Ermittlung aller erreichbaren Zustände eines Systems bei der Verwendung von ETLEBDDs präsentiert.

Ein Algorithmus für Bildberechnungen zwischen BDDs, bei dem ein neues Verfahren zur Kombination von bereits fertig berechneten Teilresultaten mit dem Ergebnis noch ausstehender Berechnungen benutzt wird, wird in Kapitel 6 beschrieben. Üblicherweise werden in Rekursionsschritten von rekursiven Algorithmen zur Kombination von BDDs zuerst Ergebnisse für Teilberechnungen ermittelt, die dann anschließend zum Resultat des Rekursionsschritts vereinigt werden. Im hier vorgestellten neuen Verfahren werden BDDs mit fertig berechneten Teilresultaten bei rekursiven Aufrufen des Algorithmus für noch ausstehende Teilberechnungen simultan mit durchlaufen und dann mit dem Ergebnis der noch ausstehenden Berechnungen vereinigt. Außerdem werden in Kapitel 6 verschiedene Strategien zur Benutzung dieses neuen Algorithmus bei der Ermittlung aller erreichbaren Zustände eines Systems angegeben. Die dynamische Symmetriereduktion (siehe Abschnitt 2.4.2.3) ist ein Ansatz zur Ausnutzung von Symmetrien bei der auf BDDs basierenden symbolischen Modellprüfung. Für deren Benutzung werden in Kapitel 7 Verbesserungen präsentiert. Dazu wird ein neuer effizienter Algorithmus zur Verwendung der symbolischen dynamischen Symmetriereduktion mit partitionierten Transitionsrelationen vorgestellt. Auch wird ein Ansatz zur zusätzlichen Ausnutzung von Zustandssymmetrien bei der dynamischen Symmetriereduktion präsentiert. Zustandssymmetrien sind innerhalb von Zuständen eines Systems vorhandene Symmetrien. Deren Ausnutzung kann zu deutlichen Laufzeitverbesserungen führen. Bisher wurde die Ausnutzung von Zustandssymmetrien nur für die explizite Modellprüfung implementiert. Der hier dafür vorgestellte Ansatz ist der erste Ansatz zu ihrer Ausnutzung bei der symbolischen Modellprüfung mit BDDs.

In Kapitel 8 werden Ergebnisse von durchgeführten Verifikationsexperimenten präsentiert. Dabei sind Ergebnisse von Verifikationsexperimenten zum Bau der Transitionsrelation und zur Erreichbarkeitsanalyse angegeben. Bei der Erreichbarkeitsanalyse werden alle von den Anfangszuständen eines Systems erreichbaren Zustände ermittelt. Die experimentellen Ergebnisse zeigen, dass alle in der vorliegenden Arbeit präsentierten neuen Ansätze zu Verbesserungen bei der Anwendung der BDD-basierten symbolischen Modellprüfung für asynchrone nebenläufige Systeme führen. Es können damit sehr große Laufzeitverringerungen und Reduktionen des benötigten Speicherbedarfs erreicht werden. Eine Zusammenfassung der Ergebnisse dieser Arbeit und ein Ausblick auf mögliche weiterführende Arbeiten ist in Kapitel 9 zu finden. Im Anhang (Kapitel 10) ist für TLEBDDs ein Algorithmus angegeben, der das in Kapitel 6 eingeführte neue Verfahren zur Berücksichtigung bereits fertig berechneter Teilresultate verwendet.

2 Übersicht einschlägiger Vorarbeiten

In diesem Kapitel wird eine Übersicht über einschlägige Vorarbeiten zu den in der vorliegenden Arbeit behandelten Themenbereichen gegeben. Dabei wird zuerst in Abschnitt 2.1 Basiswissen zur Modellprüfung, für die in dieser Arbeit Verbesserungen entwickelt wurden, präsentiert. Anschließend wird in Abschnitt 2.2 auf Binäre Entscheidungsdiagramme (engl. Binary Decision Diagrams, BDDs) eingegangen. Diese sind eine Datenstruktur, die bei der symbolischen Modellprüfung (siehe Abschnitt 2.1.4) benutzt werden kann, um Mengen symbolisch zu repräsentieren. Danach wird der in dieser Arbeit betrachtete Systemtyp der asynchronen nebenläufigen Systeme in Abschnitt 2.3 eingeführt, bevor in Abschnitt 2.4 Techniken zur Ausnutzung von in Systemen vorhandenen Symmetrien beschrieben werden. Abschließend wird in diesem Kapitel in Abschnitt 2.5 auf die Zustandsraumexploration bei Benutzung von BDDs eingegangen. Dazu werden auch Verfahren präsentiert, die Ähnlichkeiten mit den in der vorliegenden Arbeit zur Verbesserung der Zustandsraumexploration entwickelten Ansätzen aufweisen.

2.1 Modellprüfung

2.1.1 Überblick

Die Modellprüfung [53, 163] ist eine Technik zur Verifikation von Systembeschreibungen. Sie ermöglicht die vollständige Untersuchung aller Verhaltensweisen eines Systems auf die Einhaltung bestimmter gewünschter Eigenschaften. Die Verifikation selbst wird bei der Modellprüfung vollautomatisch mit Hilfe einer Software, die Modellprüfer genannt wird, durchgeführt. Als Eingabe benötigt ein Modellprüfer ein Modell des zu verifizierenden Systems und Eigenschaften, deren Gültigkeit für das System geprüft werden sollen. Das Vorgehen bei der Modellprüfung wird in der auf der nachfolgenden Seite angegebenen Abbildung 2.1 veranschaulicht. Es gibt bei der Modellprüfung drei Hauptaufgaben:

1. Die Modellierung des zu verifizierenden Systems.
2. Das Aufstellen der Eigenschaften auf deren Einhaltung das zu verifizierende System überprüft werden soll.
3. Die eigentliche Verifikation.

Bei der Modellierung muss dem Modellprüfer das zu verifizierende System in einem Formalismus übergeben werden, der als Eingabe akzeptiert wird. Dies ist bei gängigen Modellprüfern häufig eine gewöhnliche Programmiersprache (z.B. C, Java oder VHDL),

eine Teilmenge der Konstrukte einer gewöhnlichen Programmiersprache oder eine spezielle Eingabesprache zur Beschreibung von endlichen Systemen. Die Spezifikation der Eigenschaften, deren Einhaltung für das System geprüft werden soll, muss ebenfalls in einem vom Modellprüfer als Eingabe akzeptierten Formalismus erfolgen. Oft werden dazu temporale Logiken (siehe Abschnitt 2.1.3), wie zum Beispiel CTL (engl. Computation Tree Logic) [19] oder LTL (engl. Linear Temporal Logic) [160], verwendet. Temporale Logiken sind Erweiterungen von atemporalen Logiken um temporale Operatoren. Durch die Erweiterungen können mit ihnen auch Eigenschaften formuliert werden, die zeitliche Abläufe beschreiben. Bei der Formulierung der gewünschten Eigenschaften ist es wichtig, diese korrekt zu formulieren. Ebenso sollten alle wichtigen Eigenschaften, die ein System erfüllen soll, angegeben werden. Bei der Modellprüfung wird das modellierte System nur auf die Gültigkeit der verwendeten Eigenschaften überprüft. Es findet keine Prüfung der Korrektheit oder Vollständigkeit dieser Eigenschaften statt.

Nach der Übergabe des Systemmodells und der Eigenschaften an den Modellprüfer, kann dieser die Verifikation der Gültigkeit der Eigenschaften vollautomatisch durchführen. Dabei überprüft er mit Hilfe spezieller Algorithmen die Gültigkeit der Eigenschaften für alle Verhaltensweisen des Systems. Falls er herausfindet, dass das System eine Eigenschaft verletzt, gibt er ein Gegenbeispiel aus. Ein Gegenbeispiel ist ein aus aufeinander folgenden Zuständen bestehender Ausführungspfad des Systems, auf dem die Eigenschaft verletzt wird. Das Gegenbeispiel kann von einem Entwickler genutzt werden, um einen Fehler im System, einen Fehler bei der Modellierung des Systems oder einen Fehler bei der Formulierung der Eigenschaften entdecken und korrigieren zu können. Wird kein Verstoß gegen die Eigenschaften gefunden, gibt der Modellprüfer aus, dass die Verifikation erfolgreich war und das eingegebene System besitzt die gewünschten Eigenschaften.

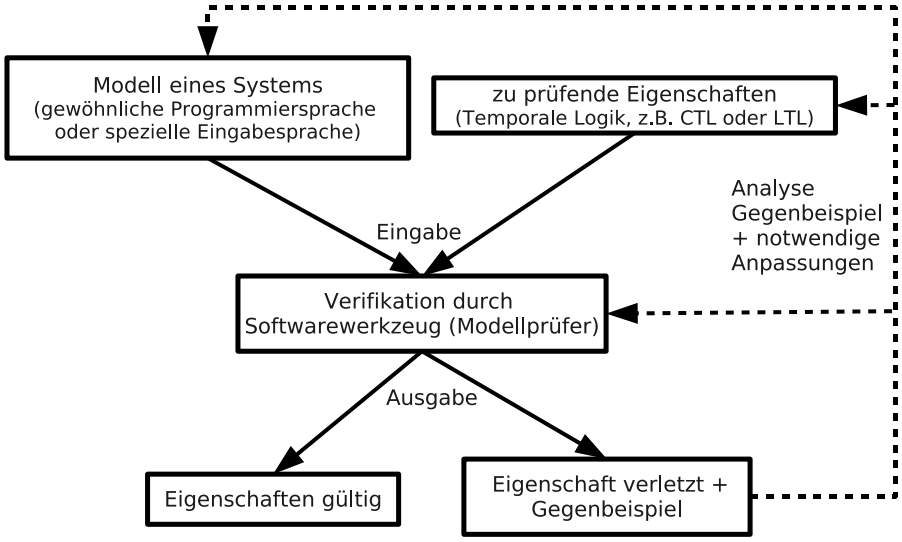

Abbildung 2.1: Veranschaulichung des Vorgehens bei der Modellprüfung.

Ein Eingreifen eines Entwicklers ist bei der Verifikation nur notwendig, um Korrekturen am eingegebenen System, an den zu prüfenden Eigenschaften oder an den Einstellungen des Modellprüfers vorzunehmen und um danach die Modellprüfung neu anzustoßen. Neben der Korrektur von Fehlern, kann ein Eingreifen eines Entwicklers bei der Verifikation notwendig sein, wenn der Modellprüfer die Verifikation des Systems aufgrund eines zu hohen benötigten Speicherbedarfs nicht vollständig durchführen kann. Zur Überprüfung aller Verhaltensweisen des Systems auf die gewünschten Eigenschaften muss der Modellprüfer alle Zustände des Systems explorieren. Die explorierten Zustände werden vom Modellprüfer gespeichert, um die Terminierung eines Verifikationslaufs entscheiden zu können. Übersteigt der dafür notwendige Speicherbedarf den Speicher des verwendeten Rechners, scheitert die Verifikation. Möglichkeiten die Verifikation des Systems dennoch zu ermöglichen sind zum Beispiel die Anwendung von Abstraktionen bei der Modellierung des Systems oder das Ändern von Parametern des Modellprüfers. Dadurch kann versucht werden den zu untersuchenden Zustandsraum zu verkleinern.

Das Problem des explodierenden Speicherbedarfs ist das Hauptproblem der Modellprüfung. Es wird als das Problem der Zustandsraumexplosion bezeichnet. Die Zahl der Zustände steigt bei der Modellprüfung exponentiell mit der Anzahl der Variablen und in nebenläufigen Systemen auch exponentiell mit der Anzahl der in einem System vorhandenen Komponenten [8]. Für nebenläufige Systeme mit mehreren Komponenten, wie sie in dieser Arbeit betrachtet werden, ergibt sich deshalb oft ein sehr großer Speicherbedarf. Dies erfordert die Entwicklung effizienter Methoden um den Speicherbedarf bei der Modellprüfung solcher Systeme zu verringern und deren Verifizierbarkeit zu ermöglichen.

2.1.2 Kripke-Strukturen

Bei der Modellprüfung prüft ein Modellprüfer nach der Eingabe eines Modells eines endlichen Systems und einer Menge von Eigenschaften die Gültigkeit dieser Eigenschaften für alle Verhaltensweisen des Systems. Ein formales Modell eines endlichen Systems, das im Bereich der Modellprüfung häufig zur Repräsentation von Systemen verwendet wird, ist die Kripke-Struktur. Zur Beschreibung von Eigenschaften von Kripke-Strukturen werden meistens temporale Logiken (siehe Abschnitt 2.1.3) verwendet.

Definition 2.1. (Kripke-Struktur, [56]) *Sei AP eine Menge atomarer Eigenschaften. Eine Kripke-Struktur M über AP ist ein Quadrupel $M = (S, S_0, R, L)$ mit:*

- *S einer endlichen Menge von Zuständen.*
- *$S_0 \subseteq S$ der Menge der Anfangszustände.*
- *$R \subseteq S \times S$ der Transitionsrelation, die total sein muss. Das heißt, $\forall s \in S\ \exists s' \in S$ mit $(s, s') \in R$.*
- *$L : S \to 2^{AP}$ einer Funktion, die jeden Zustand mit der Menge der atomaren Eigenschaften markiert, die in diesem Zustand wahr sind.*

Ein Zustand einer Kripke-Struktur entspricht dabei einer Konfiguration des modellierten Systems. Bestimmt werden Zustände durch die Werte von Variablen. Die erlaubten Zustandsübergänge werden mit der Transitionsrelation R beschrieben und Pfade aus aufeinander folgenden Zuständen modellieren Berechnungen eines Systems. Ein mit einem Zustand $s \in S$ beginnender Pfad einer Kripke-Struktur M ist eine unendliche Abfolge von Zuständen $\pi = \langle s_0, s_1, s_2, ... \rangle$, mit $s_0 = s$ und für alle $i \geq 0$ gilt $(s_i, s_{i+1}) \in R$. Dabei ist ein Zustand $s \in S$ erreichbar, wenn es einen Pfad $\langle s_0, s_1, ..., s, ... \rangle$ in M gibt, der in einem Startzustand $s_0 \in S_0$ beginnt. Eine Eigenschaft ist atomar, wenn sie sich logisch nicht in einfachere Eigenschaften zerlegen lässt. Die Funktion L ordnet jedem Zustand die Menge der atomaren Eigenschaften zu, die in ihm erfüllt sind. Für einen Zustand lassen sich die in ihm gültigen atomaren Eigenschaften aus der dem Zustand entsprechenden Konfiguration des Systems ableiten. In Abbildung 2.2 ist ein Beispiel einer Kripke-Struktur angegeben, die für zwei Komponenten ein Protokoll das den wechselseitigen Ausschluss für einen kritischen Abschnitt in einem System mit mehreren nebenläufigen Komponenten sicherstellt, modelliert.

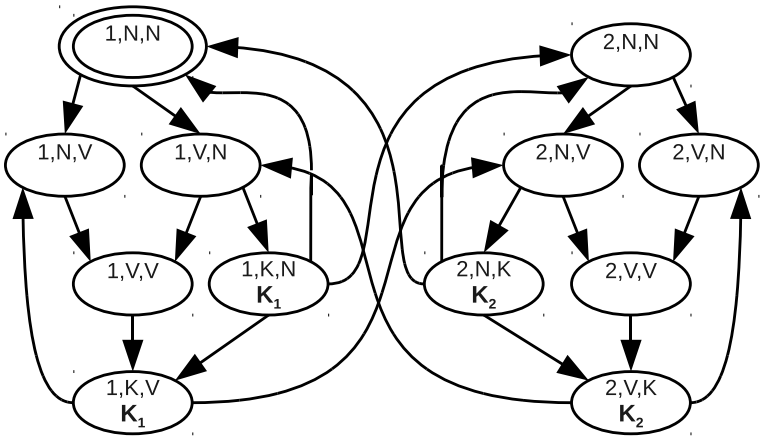

Abbildung 2.2: Kripke-Struktur für ein Protokoll, das den wechselseitigen Ausschluss für einen kritischen Abschnitt in einem nebenläufigen System mit mehreren Komponenten sicherstellt. Die Kripke-Struktur modelliert das Protokoll für zwei Komponenten.

Dieses Protokoll wird in der Literatur häufiger als Beispiel verwendet (z.B. [53, 76]) und wurde für die Verwendung in der vorliegenden Arbeit angepasst. Ein Zustandsübergangsdiagramm des Protokolls für eine Komponente K_i ($i \in \{1, ..., n\}$, $n \geq 1$) ist in Abbildung 2.3 dargestellt. Eine Komponente besitzt drei lokale Zustände: *N (neutral)*, *V (versuche)* und *K (kritisch)*. Im Zustand N wartet eine Komponente. Sie kann in den Zustand V wechseln, um zu versuchen den dem Zustand K entsprechenden kritischen Abschnitt zu betreten. Im Protokoll gibt es neben der für jede Komponente zum Speichern des lokalen Zustands der Komponente vorhandenen lokalen Variable eine globale Variable *Token*. Der Wert dieser globalen Variable ist der Index i der Komponente K_i, die den kritischen Abschnitt gerade betreten darf. Nur diese Komponente kann ihren lokalen

Zustand von V auf K ändern. Der Wert der globalen Variable selbst kann durch eine Komponente nichtdeterministisch auf den Index einer anderen vorhandenen Komponente gesetzt werden, wenn eine Komponente von ihrem Zustand K zurück in den Zustand N wechselt. Im in Abbildung 2.3 dargestellten Zustandsübergangsdiagramm ist das Abfragen bzw. Ändern des Wertes der globalen Variable *Token* durch Kantenbeschriftungen angegeben. Der wechselseitige Ausschluss wird dadurch sichergestellt, dass immer nur einer Komponente der Wechsel in ihren Zustand K erlaubt ist. Dies wird erreicht, in dem eine Komponente den kritischen Abschnitt erst betreten darf, wenn eine Komponente die globale Variable *Token* auf deren Index gesetzt hat. Das Ändern des Wertes der globalen Variable durch eine Komponente ist dabei nur beim Verlassen des kritischen Abschnitts möglich. Damit können keine zwei Komponenten gleichzeitig im kritischen Abschnitt sein. Das Protokoll funktioniert auch, wenn das angegebene Zustandsübergangsdiagramm als Übergangsdiagramm zwischen Regionen von Code, die auf die Synchronisation des Zutritts zum kritischen Abschnitt keinen Einfluss haben, angesehen wird [53].

In der Kripke-Struktur in Abbildung 2.2 entsprechen die Knoten den Zuständen der Kripke-Struktur. In den Knoten sind oben immer links der aktuelle Wert der globalen Variable und rechts daneben die Werte der lokalen Zustandsvariablen der Komponenten angegeben. Der einzige Anfangszustand $s_0 \in S_0$ ist der Zustand $s_0 = (1, N, N)$. Die in einem Zustand erfüllten atomaren Eigenschaften sind als Knotenbeschriftung unter der Variablenbelegung eines Zustands aufgeführt. Im Beispiel gibt es dafür die Knotenbeschriftung K_1 (K_2), die der atomaren Eigenschaft entspricht das Komponente 1 (Komponente 2) gerade im kritischen Abschnitt ist.

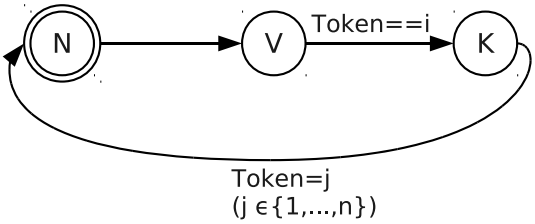

Abbildung 2.3: Zustandsübergangsdiagramm für das in Abbildung 2.2 modellierte Protokoll, das den wechselseitigen Ausschluss in einem nebenläufigen System mit n Komponenten sicherstellt, für eine Komponente K_i ($i \in \{1, ..., n\}$, $n \geq 1$).

2.1.3 Temporale Logiken

Temporale Logiken werden bei der Modellprüfung häufig verwendet, um die zu prüfenden Eigenschaften zu spezifizieren. Sie erweitern atemporale Logiken, wie z.B. die Aussagenlogik, um temporale Operatoren und erlauben das Beschreiben von Eigenschaften von Kripke-Strukturen. Mit Hilfe der temporalen Operatoren können Eigenschaften formuliert werden, die über verschiedene Zeitpunkte argumentieren und damit zeitliche Abläufe beschreiben. Es gibt verschiedene temporale Logiken, die sich in ihren Operatoren und de-

ren Bedeutung unterscheiden. Dadurch ergeben sich verschiedene Ausdrucksstärken und auch unterschiedliche Komplexitäten für die Modellprüfung mit diesen Logiken [127, 56, 8]. Im folgenden Abschnitt 2.1.3.1 wird zuerst die ausdrucksstarke Logik CTL* [53, 54, 81] beschrieben. Diese besitzt als Teilmengen die temporalen Logiken CTL [19, 53, 80] (siehe Abschnitt 2.1.3.2) und LTL [160] (siehe Abschnitt 2.1.3.3), die in praktischen Anwendungen der Modellprüfung meistens eingesetzt werden.

2.1.3.1 Verzweigende temporale Logik* (CTL*)

CTL* (engl. Computation Tree Logic*) [53, 54, 81] ist eine temporale Logik zum Beschreiben von Eigenschaften von Berechnungsbäumen. Berechnungsbäume können aus Kripke-Strukturen abgeleitet werden und beschreiben Berechnungen eines Systems. Zur Ableitung eines Berechnungsbaums aus einer Kripke-Struktur wählt man einen Anfangszustand $s \in S_0$ der Kripke-Struktur aus. Beginnend mit diesem Anfangszustand als Wurzel baut man dann einen Berechnungsbaum auf. Dies geschieht durch sukzessives Ermitteln und Anhängen von sich durch Anwenden der Transitionsrelation R der Kripke-Struktur ergebenden Nachfolgerzuständen. Dadurch erhält man einen unendlichen Baum, der alle möglichen vom gewählten Anfangszustand ausgehenden Berechnungen beinhaltet. Ein Beispiel eines Berechnungsbaums zur in Abbildung 2.2 angegebenen Kripke-Struktur ist in Abbildung 2.4 dargestellt. Der Berechnungsbaum startet mit dem Anfangszustand der Kripke-Struktur als Wurzel und zeigt alle von dort ausgehenden Berechnungen.

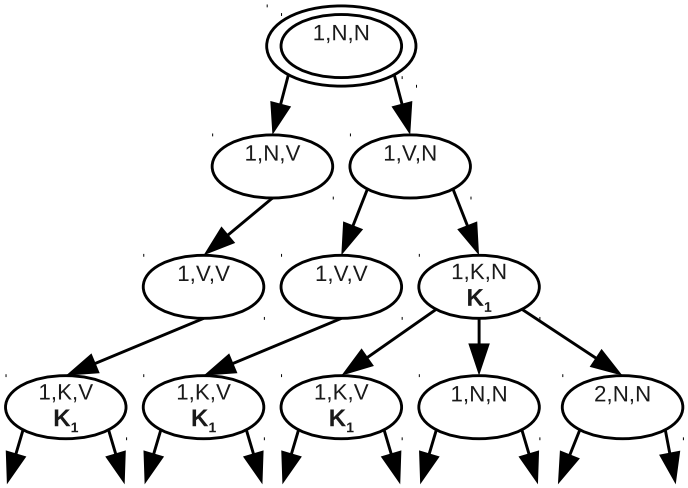

Abbildung 2.4: Berechnungsbaum zur Kripke-Struktur aus Abbildung 2.2.

CTL* ist eine Erweiterung der Aussagenlogik und verwendet neben den üblichen aussagenlogischen Operatoren zusätzlich Pfadquantoren und temporale Operatoren. Mit Hilfe der Pfadquantoren lässt sich angeben, ob Eigenschaften ausgehend von einem augenblicklichen Zustand im Berechnungsbaum für alle Folgepfade, oder für einen Folgepfad gelten müssen.

CTL* besitzt zwei Pfadquantoren:

- **A** ("*for all paths*"), Eigenschaft muss für alle Folgepfade gelten.
- **E** ("*there exists a path*"), es existiert ein Folgepfad auf dem die Eigenschaft gilt.

Temporale Operatoren werden benutzt, um Eigenschaften für einen einzelnen Folgepfad eines augenblicklichen Zustands eines Berechnungsbaums zu beschreiben. Temporale Operation von CTL* sind:

- **X** ("*next time*"), Eigenschaft gilt im unmittelbaren Nachfolgerzustand des augenblicklichen Zustands auf dem Pfad.
- **F** ("*in the future*"), Eigenschaft gilt in einem Folgezustand des Pfades.
- **G** ("*globally*"), Eigenschaft gilt in jedem Folgezustand des Pfades.
- **U** ("*until*"), kombiniert zwei Eigenschaften. Gilt, wenn irgendwann auf dem Pfad die zweite Eigenschaft gilt und auf allen vorherigen Zuständen des Pfades die erste Eigenschaft erfüllt war.

In CTL* gibt es zwei Arten von Formeln, Zustandsformeln und Pfadformeln. Zustandsformeln können in einem bestimmten Zustand wahr werden. Der Wahrheitswert einer Pfadformel entscheidet sich entlang eines gegebenen Pfades. Die Syntax von CTL*-Formeln ist wie folgt definiert:

Definition 2.2. (Syntax von CTL*, [56, 127]) *Sei AP die Menge der atomaren Eigenschaften. Sind Zustands- und Pfadformeln wie folgt definiert, dann ist jede Zustandsformel eine zulässige CTL*-Formel.*

- *Zustandsformeln:*
  - *Ist $p \in AP$, dann ist p eine Zustandsformel.*
  - *Sind f und g Zustandsformeln, so sind $\neg f$, $f \vee g$ und $f \wedge g$ ebenfalls Zustandsformeln.*
  - *Ist f eine Pfadformel, dann sind $\mathbf{E}f$ und $\mathbf{A}f$ Zustandsformeln.*
- *Pfadformeln:*
  - *Ist f eine Zustandsformel, so ist f ebenfalls eine Pfadformel.*
  - *Sind f und g Pfadformeln, so sind ebenfalls $\neg f$, $f \vee g$, $f \wedge g$, $\mathbf{X}f$, $\mathbf{F}f$, $\mathbf{G}f$ und f $\mathbf{U}$ g Pfadformeln.*

Nachdem die Syntax von CTL* und Kripke-Strukturen (siehe Abschnitt 2.1.2) definiert wurden, wird nun die Semantik von CTL* für Kripke-Strukturen angegeben. Wie in 2.1.2 eingeführt, ist ein mit einem Zustand $s \in S$ einer Kripke-Struktur startender Pfad eine unendliche Abfolge von Zuständen $\pi = \langle s_0, s_1, s_2, ... \rangle$, mit $s_0 = s$ und für alle $i \geq 0$ gilt $(s_i, s_{i+1}) \in R$. Mit π^i wird im Folgenden das Suffix $\pi = \langle s_i, s_{i+1}, ... \rangle$ eines Pfades $\pi = \langle s_0, s_1, ..., s_i, s_{i+1}, ... \rangle$ bezeichnet.

Definition 2.3. (Semantik von CTL*, [56, 127]) *Sei f eine Zustandsformel, dann bedeutet $M, s \models f$, dass f im Zustand s der Kripke-Struktur M gilt. Sei f eine Pfadformel, dann bedeutet $M, \pi \models f$, dass f entlang des Pfades π der Kripke-Struktur M gilt. Sind f_1 und f_2 Zustandsformeln sowie g_1 und g_2 Pfadformeln, so ist die Relation $\models$ wie folgt definiert:*

$$
\begin{array}{lcl}
M, s \models p & \Leftrightarrow & p \in L(s). \\
M, s \models \neg f_1 & \Leftrightarrow & M, s \not\models f_1. \\
M, s \models f_1 \vee f_2 & \Leftrightarrow & M, s \models f_1 \text{ oder } M, s \models f_2. \\
M, s \models f_1 \wedge f_2 & \Leftrightarrow & M, s \models f_1 \text{ und } M, s \models f_2. \\
M, s \models \mathbf{E}\ g_1 & \Leftrightarrow & \text{es gibt einen Pfad } \pi, \text{ der in } s \text{ beginnt, so dass gilt } M, \pi \models g_1. \\
M, s \models \mathbf{A}\ g_1 & \Leftrightarrow & \text{für jeden Pfad } \pi, \text{ der in } s \text{ beginnt, gilt } M, \pi \models g_1. \\
M, \pi \models f_1 & \Leftrightarrow & s \text{ ist der erste Zustand von } \pi \text{ und es gilt } M, s \models f_1. \\
M, \pi \models \neg g_1 & \Leftrightarrow & M, \pi \not\models g_1. \\
M, \pi \models g_1 \vee g_2 & \Leftrightarrow & M, \pi \models g_1 \text{ oder } M, \pi \models g_2. \\
M, \pi \models g_1 \wedge g_2 & \Leftrightarrow & M, \pi \models g_1 \text{ und } M, \pi \models g_2. \\
M, \pi \models \mathbf{X}\ g_1 & \Leftrightarrow & M, \pi^1 \models g_1. \\
M, \pi \models \mathbf{F}\ g_1 & \Leftrightarrow & \text{es gibt ein } k \geq 0, \text{ so dass gilt } M, \pi^k \models g_1. \\
M, \pi \models \mathbf{G}\ g_1 & \Leftrightarrow & \text{für alle } i \geq 0 \text{ gilt } M, \pi^i \models g_1. \\
M, \pi \models g_1\ \mathbf{U}\ g_2 & \Leftrightarrow & \text{es gibt ein } k \geq 0, \text{ so dass gilt } M, \pi^k \models g_2 \text{ und} \\
 & & \text{für alle } 0 \leq j < k \text{ gilt } M, \pi^j \models g_1.
\end{array}
$$

Nicht alle in diesem Abschnitt eingeführten Pfadquantoren und temporalen Operatoren werden benötigt, um die Ausdrucksstärke von CTL* zu erreichen. Es reichen zum Beispiel die Operatoren $\vee$, $\neg$, **X**, **U** und **E** um jede CTL*-Formel ausdrücken zu können [56].

Der wechselseitige Ausschluss zur Kripke-Struktur aus Abbildung 2.2 kann für einen Zustand mit den atomaren Eigenschaften $K1$ und $K2$, die erfüllt sind wenn Komponente 1 (K_1) bzw. Komponente 2 (K_2) im kritischen Abschnitt sind, durch die Eigenschaft $p = \neg(K_1 \wedge K_2)$ beschrieben werden. Erweiterte CTL*-Eigenschaften dieser Eigenschaft p sind zum Beispiel (angepasst aus [56]):

- **AG** p: Bedeutet, dass die Eigenschaft p auf jedem Pfad in jedem Zustand gilt.
- **AG** (**EF** p): Bedeutet, dass auf jedem Pfad von jedem Zustand aus immer ein Pfad erreichbar ist, auf dem die Eigenschaft p irgendwann gilt.
- **A**(**FG** p): Bedeutet, dass es auf jedem Pfad einen Zustand gibt, ab dem p auf dem Pfad immer gilt.
- **AG** (**EF** p) $\vee$ **A**(**FG** p): Es gilt die Bedeutung von **AG** (**EF** p) oder die Bedeutung von **A**(**FG** p).

2.1.3.2 Verzweigende temporale Logik (CTL)

Die temporale Logik CTL (engl. Computation Tree Logic) [19, 53, 80] ist eine Teilmenge der temporalen Logik CTL*. Sie besitzt ein verzweigendes Zeitmodell und ihre Operatoren erlauben es, wie bei CTL* Aussagen über alle Folgepfade eines gegebenen Zustands

zu machen. In CTL existiert im Gegensatz zu CTL* die Beschränkung, dass jedem temporalen Operator unmittelbar ein Pfadquantor vorangehen muss. Die Syntax von CTL unterscheidet sich von der Syntax von CTL* nur in der Definition der Pfadformeln.

Definition 2.4. (Syntax von CTL, [56, 127]) *Sei AP die Menge der atomaren Eigenschaften. Sind Zustands- und Pfadformeln wie folgt definiert, dann ist jede Zustandsformel eine zulässige CTL-Formel.*

- *Zustandsformeln (wie bei CTL*):*
  - *Ist $p \in AP$, dann ist p eine Zustandsformel.*
  - *Sind f und g Zustandsformeln, so sind $\neg f$, $f \vee g$ und $f \wedge g$ ebenfalls Zustandsformeln.*
  - *Ist f eine Pfadformel, dann sind $\mathbf{E}f$ und $\mathbf{A}f$ Zustandsformeln.*
- *Pfadformeln:*
  - *Sind f und g Zustandsformeln, so sind $\mathbf{X}f$, $\mathbf{F}f$, $\mathbf{G}f$ und f $\mathbf{U}$ g Pfadformeln.*

Beispiele für CTL-Formeln sind $\mathbf{AG}\ p$ oder $\mathbf{AG}\ (\mathbf{EF}\ p)$ (die Bedeutung der Formeln ist in Abschnitt 2.1.3.1 erklärt).

2.1.3.3 Lineare temporale Logik (LTL)

Die temporale Logik LTL (engl. Linear Temporal Logic) [160] ist ebenfalls eine Teilmenge der temporalen Logik CTL*. Anders als CTL* und CTL besitzt sie ein lineares Zeitmodell. Dadurch können mit LTL Aussagen über einzelne nicht verzweigende Berechnungspfade, aber nicht über alternative Aufspaltungen von Berechnungen, formuliert werden. Sie besitzt nur Formeln der Form $\mathbf{A}f$, denen implizit der Pfadquantor $\mathbf{A}$ vorangestellt ist und in denen f eine Pfadformel ist, die keinen weiteren Pfadquantor $\mathbf{A}$ oder $\mathbf{E}$ beinhaltet.

Definition 2.5. (Syntax von LTL, [56, 127]) *Sei AP die Menge der atomaren Eigenschaften. Dann sind folgende Pfadformeln zulässige LTL-Formeln:*

- *Ist $p \in AP$, dann ist p eine Pfadformel.*
- *Sind f und g Pfadformeln, so sind $\neg f$, $f \vee g$ und $f \wedge g$ ebenfalls Pfadformeln.*
- *Sind f und g Pfadformeln, so sind $\mathbf{X}f$, $\mathbf{F}f$, $\mathbf{G}f$ und f $\mathbf{U}$ g ebenfalls Pfadformeln.*

Beispiele für LTL-Formeln sind $\mathbf{AG}\ p$ oder $\mathbf{A}(\mathbf{FG}\ p)$ (die Bedeutung der Formeln ist in Abschnitt 2.1.3.1 erklärt).

2.1.3.4 Vergleich der temporalen Logiken CTL*, CTL und LTL

In diesem Abschnitt wird die Ausdrucksstärke der temporalen Logiken CTL*, CTL und LTL verglichen. Eine temporale Logik ist dabei hier mächtiger als eine andere, wenn man mit ihr Eigenschaften ausdrücken kann, die mit der anderen nicht beschrieben werden können. In [52, 81, 130] wurde gezeigt, dass die eingeführten temporalen Logiken verschiedene

Ausdrucksmächtigkeiten haben. Der Vergleich der drei temporalen Logiken ist in Abbildung 2.5 mit der Angabe von Beispieleigenschaften aus Abschnitt 2.1.3.1 veranschaulicht. Am mächtigsten ist die temporale Logik CTL*. Da CTL und LTL Teilmengen dieser Logik sind, können alle Eigenschaften dieser Logiken mit CTL* beschrieben werden. Vergleicht man die temporalen Logiken CTL und LTL, dann existieren CTL-Formeln die nicht mit LTL formuliert werden können (z.B. die CTL-Formel **AG**(**EF** p) aus Abbildung 2.5), sowie auch LTL-Formeln die nicht mit CTL formuliert werden können (z.B. die LTL-Formel **A**(**FG** p)). Ebenso gibt es Formeln, die mit beiden temporalen Logiken ausgedrückt werden können (z.B. **AG** p). Daher ist keine dieser beiden Logiken ausdrucksstärker als die andere.

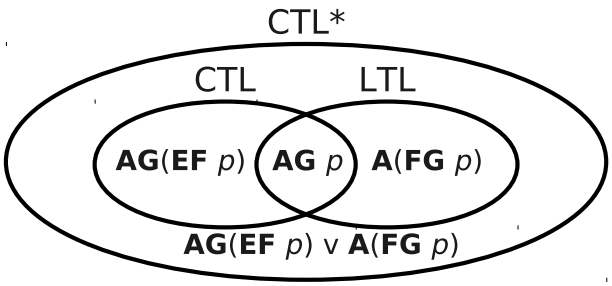

Abbildung 2.5: Vergleich der Ausdrucksmächtigkeiten der temporalen Logiken CTL*, CTL und LTL mit Beispieleigenschaften aus Abschnitt 2.1.3.1.

2.1.4 Explizite und symbolische Modellprüfung

Bei der expliziten Modellprüfung wird eine explizite Repräsentation von Kripke-Strukturen verwendet und Zustände werden explizit abgespeichert. Eine Kripke-Struktur ist im wesentlichen ein Graph, in dem die Knoten den Zuständen und die Kanten den Transitionen entsprechen. Daher basierten die ersten Algorithmen zur Modellprüfung auf einer Repräsentation der Kripke-Struktur als markiertem gerichteten Graphen [53, 54, 163].

Es werden bei der expliziten Modellprüfung hauptsächlich zwei Ansätze verfolgt [21]. Die ursprünglichen Algorithmen dazu bauten zuerst die Kripke-Struktur des zu verifizierenden Systems vollständig auf. Danach wurde die Gültigkeit der angegebenen Eigenschaften für die generierte Kripke-Struktur überprüft. Häufig ist aber zur Überprüfung von Eigenschaften und besonders zur Entdeckung nicht erfüllter Eigenschaften die Exploration des gesamten Zustandsraums nicht notwendig. Deshalb wurden Techniken entwickelt, bei denen die Überprüfung von Eigenschaften während des Aufbaus der Kripke-Struktur durchgeführt wird. Falls die Exploration der gesamten Kripke-Struktur für die Überprüfung von Eigenschaften nicht nötig ist, kann die Exploration der Kripke-Struktur dadurch auf die für die Eigenschaften relevanten Teile beschränkt werden, wodurch die zu speichernde Zustandszahl verringert werden kann. Entsprechende Algorithmen für die explizite Modellprüfung werden auch *on-the-fly Algorithmen* genannt (siehe z.B. [62, 86, 87, 88]). Aufgrund des Problems der Zustandsraumexplosion (siehe Abschnitt 2.1.1) besitzen zu

verifizierende Systeme häufig eine sehr große Anzahl an Zuständen [56]. Beim expliziten Abspeichern von Zuständen kann der dafür nötige Speicherbedarf dadurch oft den vorhandenen Speicherplatz übersteigen, wodurch die explizite Modellprüfung nicht mehr erfolgreich durchgeführt werden kann. Häufig ist die Anzahl der zu speichernden Zustände auch bei Verwendung von Ansätzen zur Verringerung des Speicherbedarfs (wie z.B. die Benutzung von *on-the-fly Algorithmen* oder der Reduktion des Zustandsraums durch Halbordnung (engl. partial order reduction, siehe z.B. [8, 56, 112, 157])) immer noch sehr groß.

Eine Alternative, die sich in der Praxis als sehr effizient erwiesen hat und die das Problem der Zustandsraumexplosion oft abmildert, ist die symbolische Modellprüfung [38, 142]. Bei der symbolischen Modellprüfung werden Zustandsmengen und die Transitionsrelation symbolisch repräsentiert und es wird mit Mengen von Zuständen und Transitionen gearbeitet. Statt einzelnen Zuständen und Transitionen, wie bei der expliziten Modellprüfung, werden Mengen von Zuständen und Mengen von Transitionen repräsentiert. Der prominenteste Ansatz dazu, der in Abschnitt 2.1.5 vorgestellt wird, benutzt eine binäre Kodierung von Zuständen. Dadurch wird die Kodierung von Mengen von Zuständen und der Transitionsrelation durch Boolesche Funktionen, den charakteristischen Funktionen der repräsentierten Mengen, möglich.

Es gibt verschiedene Datenstrukturen um Boolesche Funktionen zu repräsentieren. Eine bei der symbolischen Modellprüfung häufig verwendete Datenstruktur, mit der Systeme mit sehr großer Zustandszahl verifiziert werden können, sind BDDs (siehe Abschnitt 2.2). BDDs sind zur Modellprüfung sehr gut geeignet, da sie Mengen oft kompakt repräsentieren können. Außerdem gibt es effiziente Implementierungen von in Algorithmen für die symbolische Modellprüfung häufig verwendeten Operationen (z.B. Vereinigung oder Durchschnitt von Mengen). Ein weiterer Ansatz zur symbolischen Modellprüfung basiert auf der Aussagenlogik. Hier werden Boolesche Funktionen durch konjunktive Normalformen repräsentiert und diese zur Modellierung des Verifikationsproblems verwendet. Ein Beispiel für den Einsatz dieses Ansatzes ist die beschränkte Modellprüfung (engl. Bounded Model Checking, BMC) [23].

2.1.5 Symbolische Repräsentation von Kripke-Strukturen

Wie in Abschnitt 2.1.4 beschrieben, wird bei der symbolischen Modellprüfung mit Mengen von Zuständen und mit Mengen von Transitionen gearbeitet. Der bekannteste und in dieser Arbeit verwendete Ansatz zur symbolischen Modellprüfung benutzt eine binäre Kodierung von Zuständen [8] und eine Repräsentation von Mengen durch Boolesche Funktionen.

Definition 2.6. (Boolesche Funktion, [127]) *Eine Boolesche Funktion ist eine Abbildung* $f : \mathbb{B}^n \to \mathbb{B}$*, wobei* $\mathbb{B} = \{0, 1\}$.

Für eine endliche Kripke-Struktur $M = (S, S_0, R, L)$ kann eine binäre Kodierung von Zuständen mit Bitvektoren der Länge $n \geq \lceil log_2|S| \rceil$ erfolgen[1]. Den Zuständen können dann binäre Kodierungen durch eine beliebige injektive Abbildung $kodiere : S \to \mathbb{B}^n$

[1] Es wird dabei angenommen das $|S| \geq 2$, da Kripke-Strukturen in der Praxis mehrere Zustände besitzen.

zugeordnet werden. Um anzugeben, ob Elemente in einer Teilmenge einer übergeordneten Grundmenge enthalten sind, können charakteristische Funktionen verwendet werden.

Definition 2.7. (Charakteristische Funktion, [127]) *Es seien A und C Mengen mit $C \subseteq A$. Die charakteristische Funktion $\chi_C : A \to \mathbb{B}$ ist definiert als*

$$\chi_C(x) = \begin{cases} 1 & , \textit{falls } x \in C \\ 0 & , \textit{falls } x \notin C \end{cases}$$

Charakteristische Funktionen können daher benutzt werden, um Teilmengen einer Grundmenge darzustellen. Mengen $C \subseteq S$ von Zuständen einer Kripke-Struktur können gemäß ihrer binären Kodierung nach der Funktion *kodiere* durch charakteristische Funktionen $\chi_C(x) : \mathbb{B}^n \to \mathbb{B}$ repräsentiert werden. $\chi_C(x)$ wertet genau für die Eingabevektoren $x \in \mathbb{B}^n$ auf 1 aus, die Kodierungen von Zuständen der Menge C entsprechen. Solche charakteristischen Funktionen sind Boolesche Funktionen. Diese werden bei der symbolischen Modellprüfung symbolisch, mit Datenstrukturen wie z.B. BDDs (siehe Abschnitt 2.2), repräsentiert. Falls die benutzte Datenstruktur Mengenoperationen effizient unterstützt, wie es z.B. bei BDDs der Fall ist, können diese zur effizienten Manipulation der repräsentierten Mengen bei der Modellprüfung eingesetzt werden.

Die Transitionsrelationen $R \subseteq S$ x S von Kripke-Strukturen können ebenso durch charakteristische Funktionen dargestellt werden. Transitionen besitzen einen Ausgangszustand und einen Nachfolgerzustand. Eine Transition kann binär durch die Kombination der binären Kodierungen des Ausgangs- und des Nachfolgerzustands kodiert werden. Daher besitzen charakteristische Funktionen zur Darstellung von Mengen von Transitionen die doppelte Anzahl an Eingabevariablen, nämlich n-Variablen für den Ausgangszustand und n-Variablen für den Nachfolgerzustand. Die Transitionsrelationen können daher durch charakteristische Funktionen $\chi_R(x) : \mathbb{B}^{2n} \to \mathbb{B}$ repräsentiert werden. $\chi_R(x)$ ordnet Zustandspaaren $(s, s') \in R$, zwischen denen ein Zustandsübergang möglich ist, den Funktionswert 1 zu. Vorhandene atomare Eigenschaften $p \in AP$ können durch charakteristische Funktionen für Zustandsmengen $\{s | p \in L(s)\}$, die aus Zuständen in denen die Eigenschaft erfüllt ist bestehen, repräsentiert werden.

2.1.6 Algorithmen zur symbolischen Modellprüfung

In diesem Abschnitt werden Algorithmen für die symbolische Modellprüfung mit den temporalen Logiken CTL (siehe Abschnitt 2.1.6.1) und LTL (siehe Abschnitt 2.1.6.2) beschrieben. Die Modellprüfung von CTL*-Formeln kann durch eine Kombination von Algorithmen zur LTL- und CTL-Modellprüfung durchgeführt werden [83]. In der Praxis verwenden die meisten Modellprüfer aber die temporalen Logiken CTL und LTL und es werden üblicherweise nur CTL-Formeln oder nur LTL-Formeln benutzt. Daher werden hier nur Algorithmen zur Modellprüfung mit den temporalen Logiken CTL und LTL vorgestellt. Üblicherweise ist man bei der Modellprüfung daran interessiert, ob eine in einer temporalen Logik angegebene Formel ϕ in den Anfangszuständen einer gegebenen Kripke-Struktur (siehe Abschnitt 2.1.2) erfüllt ist (d.h. ob für eine Kripke-Struktur $M = (S, S_0, R, L)$ gilt $\forall s \in S_0 : M, s \models \phi$). Die dazu nötigen Algorithmen unterscheiden sich je nach gewählter temporaler Logik.

Neben der Modellprüfung von temporallogischen Formeln wird die Erreichbarkeitsanalyse, von der die symbolische Erreichbarkeitsanalyse in Abschnitt 2.1.6.3 beschrieben wird, häufig bei der Modellprüfung eingesetzt. Die symbolische Erreichbarkeitsanalyse wurde auch bei den für die vorliegende Arbeit durchgeführten Verifikationsexperimenten benutzt. Da sich die vorliegende Arbeit mit der symbolischen Modellprüfung mit BDDs [33] (siehe Abschnitt 2.2) befasst, werden in den nachfolgenden Abschnitten nur Algorithmen dafür beschrieben. Informationen zu den ursprünglichen Algorithmen für die explizite Modellprüfung sind in Abschnitt 2.1.4 zu finden und weitere Referenzen zur expliziten Modellprüfung sind in den Abschnitten 2.1.6.1 und 2.1.6.2 angegeben.

2.1.6.1 Symbolische CTL-Modellprüfung

In diesem Abschnitt wird ein effizienter Algorithmus zur symbolischen Modellprüfung mit CTL-Formeln, der auch in [56, 127] vorgestellt wurde, angegeben. Der Algorithmus verwendet Fixpunktcharakterisierungen der CTL-Operatoren. Für eine gegebene endliche Kripke-Struktur bildet die Potenzmenge 2^S der Zustandsmenge S einer Kripke-Struktur unter der Teilmengenbeziehung ($\subseteq$) als Ordnungsrelation einen vollständigen Verband V: $V = (2^S, \subseteq)$. Das kleinste Element des Verbands ist die leere Menge ($\emptyset$) und das größte Element ist die gesamte Zustandsmenge (S). Die Fixpunktcharakterisierungen der CTL-Operatoren basieren auf Funktionen $g : 2^S \rightarrow 2^S$, die Mengen von Zuständen wieder auf Mengen von Zuständen abbilden. Für eine solche Funktion g die außerdem monoton ist, kann gezeigt werden, dass die Funktion einen kleinsten Fixpunkt $fp_{min}(g)$ und einen größten Fixpunkt $fp_{max}(g)$ besitzt.

Die folgenden Definitionen werden für den für diese Arbeit wichtigen endlichen Mengenverband $(2^S, \subseteq)$ angegeben. Dieser Verband besitzt als Ordnungsrelation die Teilmengenbeziehung ($\subseteq$), für die Verbandsoperation Infimum die Mengenoperation Durchschnitt "$\cap$" und für die Verbandsoperation Supremum die Mengenoperation Vereinigung "$\cup$". Allgemeinere Formulierungen sind in [127, 174] zu finden.

Definition 2.8. (Monotone Funktion, [56]) *Eine Funktion $g : 2^S \rightarrow 2^S$ heißt monoton, wenn für Teilmengen $S_1, S_2 \subseteq S$ mit $S_1 \subseteq S_2$ auch gilt $g(S_1) \subseteq g(S_2)$.*

Definition 2.9. (Fixpunkt) *Sei $g : 2^S \rightarrow 2^S$ eine Funktion, dann ist eine Teilmenge $S_1 \subseteq S$ ein Fixpunkt von g, wenn gilt $g(S_1) = S_1$.*

- *S_1 heißt kleinster Fixpunkt von g, wenn für jeden Fixpunkt S_2 von g gilt $S_1 \subseteq S_2$.*
- *S_1 heißt größter Fixpunkt von g, wenn für jeden Fixpunkt S_2 von g gilt $S_2 \subseteq S_1$.*

Eine monotone Funktion $g : 2^S \rightarrow 2^S$ ist bei endlicher Zustandsmenge S auch vereinigungs- und schnittstetig [56].

Definition 2.10. (Vereinigungsstetige Funktion, [56]) *Eine Funktion $g : 2^S \rightarrow 2^S$ heißt vereinigungsstetig, wenn für eine unendliche Folge S_1, S_2, S_3,... von Elementen $S_i \in 2^S$ mit $S_1 \subseteq S_2 \subseteq S_3$... gilt: $\bigcup_i g(S_i) = g(\bigcup_i S_i)$.*

Definition 2.11. (Schnittstetige Funktion, [56]) *Eine Funktion $g : 2^S \to 2^S$ heißt schnittstetig, wenn für eine unendliche Folge S_1, S_2, S_3,... von Elementen $S_i \in 2^S$ mit $S_1 \supseteq S_2 \supseteq S_3$... gilt: $\bigcap_i g(S_i) = g(\bigcap_i S_i)$.*

Im Folgenden wird durch $g^i(Z)$ die i-fache Anwendung der Funktion g auf eine Zustandsmenge $Z \in 2^S$ beschrieben, wobei $g^0(Z) = Z$ und $g^{i+1}(Z) = g(g^i(Z))$. Satz 2.12 besagt, dass der kleinste und der größte Fixpunkt monotoner Funktionen g über vollständigen Verbänden eindeutig ist.

Satz 2.12. (Fixpunktsatz, [127, 131, 174, 183]) *Sei S eine Menge, $(2^S, \subseteq)$ ein vollständiger Verband und $g : 2^S \to 2^S$ eine monotone Funktion. Dann hat g einen eindeutig bestimmten kleinsten Fixpunkt $fp_{min}(g) = \cap\{Z | g(Z) \subseteq Z\}$ und einen eindeutig bestimmten größten Fixpunkt $fp_{max}(g) = \cup\{Z | g(Z) \supseteq Z\}$. Ist g vereinigungsstetig dann gilt $fp_{min}(g) = \bigcup_{i\in\mathbb{N}} g^i(\emptyset)$ und ist g schnittstetig dann gilt $fp_{max}(g) = \bigcap_{i\in\mathbb{N}} g^i(S)$.*

Für den endlichen Mengenverband $(2^S, \subseteq)$ und monotone Funktionen g werden die Fixpunkte $fp_{min}(g) = \bigcup_{i\in\mathbb{N}} g^i(\emptyset)$ bzw. $fp_{max}(g) = \bigcap_{i\in\mathbb{N}} g^i(S)$ durch endliche iterative Anwendung von g erreicht und es gibt ein $i_0 \in \{1, ..., |S|\}$ mit $fp_{min}(g) = g^{i_0}(\emptyset)$ und ein $j_0 \in \{1, ..., |S|\}$ mit $fp_{max}(g) = g^{j_0}(S)$ [56]. Der kleinste und größte Fixpunkt können also ausgehend von der leeren Menge, bzw. der gesamten Menge S, iterativ in endlich vielen Schritten berechnet werden.

Wird bei gegebener Kripke-Struktur M jeder CTL-Formel f die Menge aller Zustände $s \in S$ der Kripke-Struktur zugeordnet, in denen f gilt $(\{s | M, s \models f\})$, dann lassen sich die CTL-Basisoperatoren durch Fixpunktcharakterisierungen beschreiben [80]:

$$\begin{aligned}
\mathbf{AF}f_1 &= fp_{min}(Z, f_1 \vee \mathbf{AX}Z) \\
\mathbf{EF}f_1 &= fp_{min}(Z, f_1 \vee \mathbf{EX}Z) \\
\mathbf{AG}f_1 &= fp_{max}(Z, f_1 \wedge \mathbf{AX}Z) \\
\mathbf{EG}f_1 &= fp_{max}(Z, f_1 \wedge \mathbf{EX}Z) \\
\mathbf{A}(f_1\mathbf{U}f_2) &= fp_{min}(Z, f_2 \vee (f_1 \wedge \mathbf{AX}Z)) \\
\mathbf{E}(f_1\mathbf{U}f_2) &= fp_{min}(Z, f_2 \vee (f_1 \wedge \mathbf{EX}Z))
\end{aligned} \tag{2.1}$$

In den Fixpunktcharakterisierungen von 2.1 wird die Bedeutung der CTL-Operatoren im zweiten Argument der Funktionen $fp_{min}()$ bzw. $fp_{max}()$ angegeben. Dies entspricht der Berechnungsvorschrift (Funktion g), die bei der Fixpunktberechnung in jeder Iteration angewandt wird. Für die Operatoren EX und AX sind keine Fixpunktcharakterisierungen nötig, da sie nur Aussagen über direkte Nachfolgerzustände machen und damit direkt berechnet werden können. Kleinste Fixpunkte werden für CTL-Operatoren verwendet, die ausdrücken das eine Bedingung irgendwann gelten muss. Die Berechnung der Zustände beginnt für diese Operatoren mit der Anwendung der Iterationsvorschrift g auf die leere Menge $(fp_{min}(\emptyset, g))$. Größte Fixpunkte werden für Eigenschaften verwendet, die immer gelten sollen. Die Berechnung der entsprechenden Zustandsmengen erfolgt dort ausgehend von der gesamten Zustandsmenge S $(fp_{max}(S, g))$. Für die CTL-Operatoren **EG** und **EU**

finden sich in [56, 127] Beweise, in denen gezeigt wird, dass die dafür angegebenen Iterationsvorschriften monoton sind und die Fixpunktcharakterisierungen den gewünschten Fixpunkt liefern.

Mit Hilfe der Fixpunktcharakterisierungen der CTL-Operatoren kann durch Berechnung der entsprechenden Fixpunkte für eine Kripke-Struktur die Menge der Zustände $s \in S$ berechnet werden, in denen eine CTL-Formel f gilt ($\{s|M, s \models f\}$). In der Regel interessiert man sich bei der Modellprüfung dafür, ob eine gegebene Eigenschaft f in den Anfangszuständen $S_0 \in S$ einer Kripke-Struktur M gilt. Um herauszufinden, ob eine CTL-Formel in den Anfangszuständen gilt, reicht es zu überprüfen ob alle Anfangszustände in der Menge der Zustände, in denen die CTL-Formel gilt, enthalten sind. Das Berechnen der Zustände einer Kripke-Struktur, die eine gegebene CTL-Formel erfüllen, kann durch Aufspalten der CTL-Formel in Teilformeln erfolgen. Dabei wird beginnend mit den atomaren Propositionen sukzessive für alle Teilformeln die Menge der Zustände ermittelt, in denen die Teilformeln erfüllt sind. Dies wird durchgeführt, bis schließlich die Zustandsmenge ermittelt wurde, die die gesamte Formel erfüllt. Dies kann mit folgenden Operationen durchgeführt werden [127]:

- Auswertung der atomaren Eigenschaften von Zuständen ($L(s)$).
- Mengenoperationen für die aussagenlogischen Operatoren.
- Auswertung der Transitionsrelation für den *next time*($\mathbf{X}$) Operator.
- Fixpunktberechnung gemäß Fixpunktcharakterisierungen von CTL-Operatoren.

Die symbolische CTL-Modellprüfung kann nun unter Benutzung der Fixpunktcharakterisierungen (bzw. Durchführung der entsprechenden Fixpunktberechnungen) mit Datenstrukturen durchgeführt werden, die die Repräsentation von Mengen von Zuständen erlauben und die zuvor genannten Operationen unterstützen. Eine solche Datenstruktur sind z.B. die in dieser Arbeit zur symbolischen Repräsentation von Mengen verwendeten BDDs (siehe Abschnitt 2.2). Diese ermöglichen häufig eine kompakte Repräsentation von Mengen. Außerdem gibt es effiziente Implementierungen für die bei der symbolischen Modellprüfung notwendigen Operationen.

Der fixpunktbasierte Algorithmus zur symbolischen CTL-Modellprüfung hat polynomielle Komplexität bezüglich Modellgröße $|M|$ und Länge der temporallogischen Formel $|f|$ [127]. Die Komplexität ist dabei bezüglich der Anzahl der bei den Fixpunktberechnungen verwendeten Operationen angegeben. Bei symbolischen Repräsentationen können die Datenstrukturen für manche Mengen aber exponentielle Größe besitzen. Dadurch kann wieder exponentieller Aufwand entstehen. Dies ist zum Beispiel bei der Verwendung von BDDs möglich, für die der Aufwand zur Berechnung Boolescher Operationen zwischen BDDs polynomiell bezüglich ihrer Größe ist.

2.1.6.2 Symbolische LTL-Modellprüfung

Für die Modellprüfung von LTL-Formeln werden hauptsächlich zwei verschiedene Ansätze verwendet. Beide basieren darauf eine Produktstruktur zu bauen. Dies geschieht

durch Komposition einer Struktur, die das zu verifizierende System repräsentiert und einer Struktur, die die zu überprüfende LTL-Eigenschaft repräsentiert. Die Gültigkeit der zu prüfenden LTL-Formel wird für ein gegebenes System dann mit Hilfe der Produktstruktur entschieden.

Ein automatentheoretischer Ansatz dazu wurde in [187] vorgestellt. Dieser verwendet Büchi-Automaten und entscheidet die Gültigkeit einer LTL-Formel über die von einem konstruierten Produktautomaten akzeptierte Sprache. In [138] wurde ein als Tableau-Methode bezeichneter Ansatz vorgestellt. Er basiert auf einem Tableau, das für die zu verifizierende temporale Eigenschaft gebaut wird. Beide Ansätze besitzen lineare Komplexität bezüglich Modellgröße $|M|$ und exponentielle Komplexität bezüglich der Länge der zu prüfenden temporallogischen Formel $|f|$. Für kleine Formellängen sind diese effiziente Algorithmen für die LTL-Modellprüfung. Dies ist in der Praxis oft der Fall, da die Länge der temporallogischen Formel häufig deutlich kleiner ist als die Modellgröße [127].

Eine Verbesserung der Tableau-Methode aus [138] zur Anwendung für die symbolische Modellprüfung wurde in [51] präsentiert. Die dort präsentierte Methode wird in diesem Abschnitt vorgestellt und von den meisten symbolischen Modellprüfern zur Verifikation von LTL-Formeln benutzt [171]. Der Aufbau von LTL-Formeln wurde in Abschnitt 2.1.3.3 beschrieben. LTL-Formeln sind von der Form $\mathbf{A}\ f$, denen implizit der Pfadquantor $\mathbf{A}$ vorangestellt ist. Gesucht ist die Zustandsmenge $s \in S$ einer Kripke-Struktur M, für die $M, s \models \mathbf{A}f$ gilt. In den Zuständen $s \in S$ einer Kripke-Struktur in denen $M, s \models \mathbf{A}f$ gilt, ist auch $M, s \models \neg\mathbf{E}\neg f$ gültig. Deshalb reicht es aus Formeln der Form $\neg\mathbf{E}\neg f$ prüfen zu können. Die Tableau-Methode nutzt dies aus und überprüft Formeln der Form $\mathbf{E}f$. Dabei bezeichnet f hier in der Formel $\mathbf{E}f$ und im Rest dieses Abschnitts schon die Negierung der eigentlich zu prüfenden LTL-Eigenschaft. Damit können Zustände der Kripke-Struktur M gefunden werden, von denen ein Pfad ausgeht, der die eigentlich zu prüfende Eigenschaft verletzt. Dazu wird bei gegebener Kripke-Struktur M und temporallogischer Formel f zuerst ein Tableau T für die Pfadformel f erzeugt. T ist eine Kripke-Struktur $T = (S_T, R_T, L_T)$ in der jeder mögliche Pfad, der die Formel f erfüllt, enthalten ist. Zustände dieser Kripke-Struktur entsprechen Mengen von Teilformeln der zu prüfenden LTL-Eigenschaft. Aus Platzgründen wird hier nicht weiter auf den Aufbau eines Tableaus für eine LTL-Formel eingegangen. Weitere Informationen dazu finden sich in [51]. Die Kripke-Struktur, des auf die LTL-Formel zu verifizierenden Systems, enthält dagegen die Menge der für das System möglichen Pfade. Aus diesen beiden Kripke-Strukturen wird eine Produktstruktur P erzeugt (siehe Definition 2.15), die alle Pfade enthält, die in T und in M erlaubt sind. Die Kripke-Struktur M und das Tableau T können unterschiedliche Mengen von atomaren Eigenschaften besitzen, da im Tableau nur atomare Eigenschaften AP_T der damit repräsentierten LTL-Formel vorhanden sind. Daher muss beim Argumentieren über Pfade beider Strukturen eine Einschränkung auf die im Tableau vorhandenen atomaren Eigenschaften erfolgen. Dazu wird eine erweiterte Markierungsfunktion für Pfade eingeführt.

Definition 2.13. (Erweiterte Markierungsfunktion $\bar{L}$, [127]) *Gegeben sei ein Pfad $\pi = \langle s_0, s_1, ...\rangle$ einer Kripke-Struktur M. Die Markierungsfunktion $\bar{L}$ für Pfade ist definiert als $\bar{L}(\pi) = \bar{L}(\langle s_0, s_1, ...\rangle) = \langle L(s_0), L(s_1), ...\rangle$.*

Nun kann die Gültigkeit des nachfolgenden Satzes 2.14 gezeigt werden, in dem die erweiterte Markierungsfunktion benutzt wird. Mit $Sat(f)$ wird im Satz die Menge der Zustände des Tableaus bezeichnet, in denen eine Eigenschaft f erfüllt ist.

Satz 2.14. ([127]) *Gegeben sei ein Tableau T für eine Pfadformel f mit den atomaren Eigenschaften AP_T. Für jede Kripke-Struktur M und jeden Pfad $\pi_M = \langle s_M^0, s_M^1, ...\rangle$ von M gilt, wenn $M, \pi_M \models f$ gilt, dann gibt es auch einen Pfad $\pi_T = \langle s_T^0, s_T^1, ...\rangle$ in T, der in einem Zustand s_T^0 aus $Sat(f)$ startet, mit $\langle L_M(s_M^0) \cap AP_T, L_M(s_M^1) \cap AP_T, ...\rangle = \bar{L}_T(\pi_T)$.*

Damit gibt es bei Gültigkeit einer Pfadformel f für einen Pfad einer Kripke-Struktur M einen Pfad im Tableau T zur Formel, der in einem Zustand von T startet, in dem f erfüllt ist. Die Gültigkeit von LTL-Formeln für eine Kripke-Struktur M kann daher durch Pfade, die sowohl in M als auch in T vorhanden sind, entschieden werden. Dazu wird bei der Tableau-Methode eine Produktstruktur durch Komposition von Kripke-Struktur M und Tableau T konstruiert.

Definition 2.15. (Produktstruktur, [127]) *Gegeben seien eine Kripke-Struktur $M = (S_M, R_M, L_M)$ und ein Tableau $T = (S_T, R_T, L_T)$. Die Produktstruktur P ist definiert als $P = (S_P, R_P, L_P)$, mit*

- $S_P = \{(s_T, s_M) | s_T \in S_T,\ s_M \in S_M$ *und* $L_M(s_M) \cap AP_T = L_T(s_T)\}$.
- $R_P((s_T, s_M), (s'_T, s'_M)) \Leftrightarrow R_T(s_T, s'_T)$ *und* $R_M(s_M, s'_M)$.
- $L_P(s_T, s_M) = L_T(s_T)$.

Die Produktstruktur besitzt als Pfadmenge die Menge der Pfade, die in M und in T möglich sind. Die Gültigkeit von LTL-Formeln kann nun durch das Finden von unendlichen Pfaden in der Produktstruktur entschieden werden. Allerdings muss bei Formeln, bei denen eine Eigenschaft irgendwann erfüllt sein muss (z.B. bei den temporalen Operatoren **F** oder **U**), noch sichergestellt werden, dass die Eigenschaft irgendwann erfüllt ist (siehe Abschnitt 2.1.3.1). Dies wird in [51] durch Anwendung der CTL-Modellprüfung mit Fairnessbedingungen durchgeführt. Da der **F**-Operator durch den **U**-Operator ausgedrückt werden kann, reicht es dabei den **U**-Operator zu betrachten. Dazu wird für die Produktstruktur die Menge $Sat(f)$ der Zustände der Produktstruktur, die eine temporallogische Formel f erfüllen, definiert durch: $(s_T, s_M) \in Sat(f) \Leftrightarrow s_T \in Sat(f)$ *im Tableau T*. Unter Verwendung der CTL-Modellprüfung mit den Fairnessbedingungen

$$\{Sat(\neg(f_1 \mathbf{U} f_2) \vee f_2) \,|f_1 \,\mathbf{U} f_2 \text{ kommt in } f \text{ vor}\}. \tag{2.2}$$

wird nun die Menge der Zustände $S_f \subseteq Sat(f)$ der Produktstruktur bestimmt, in denen die CTL-Formel $\mathbf{EG}\ TRUE$ erfüllt ist. Bei der CTL-Modellprüfung mit Fairness [56, 127] wird die Gültigkeit der angegebenen Eigenschaften nur für *faire Pfade* geprüft. Es wird eine Menge von Fairnessbedingungen angegeben, die jeweils den Zustandsmengen entsprechen, in denen die Fairnessbedingungen erfüllt sind. Faire Pfade sind solche Pfade, auf

denen jede Fairnessbedingung unendlich oft erfüllt ist.[1] Das Ergebnis der Modellprüfung ist die Menge aller Zustände der Produktstruktur, von denen unendliche Pfade ausgehen, die die Formel **EG** $TRUE$ und die Fairnessbedingungen erfüllen. Dies entspricht den Zuständen $s \in S_M$ für die $M, s \models \mathbf{E}f$ gilt. Ist diese Zustandsmenge nicht leer, dann wurden Zustände gefunden, in denen die eigentlich zu prüfende unnegierte LTL-Eigenschaft für die Kripke-Struktur M nicht gilt. Für die CTL-Modellprüfung mit Fairness lassen sich erweiterte Fixpunktcharakterisierungen der CTL-Operatoren (Fixpunktcharakterisierungen für CTL ohne Fairness siehe Abschnitt 2.1.6.1) unter Berücksichtigung der Fairnessbedingungen angeben [56, 127]. Die beschriebene Tableau-Methode für die symbolische Modellprüfung kann nun mit Datenstrukturen, die die Manipulation von Mengen erlauben, durchgeführt werden (z.B. mit BDDs (siehe Abschnitt 2.2)).

2.1.6.3 Symbolische Erreichbarkeitsanalyse

In diesem Abschnitt wird die symbolische Erreichbarkeitsanalyse beschrieben. Diese arbeitet mit Mengen von Zuständen, die mit symbolischen Datenstrukturen (siehe auch Abschnitt 2.1.4), wie z.B. BDDs, repräsentiert werden. Bei gegebener Zustandsmenge Z und Kripke-Struktur $M = (S, S_0, R, L)$ werden bei der Erreichbarkeitsanalyse alle von der Zustandsmenge Z erreichbaren Zustände bestimmt. Es gibt zwei Varianten der Erreichbarkeitsanalyse, die Vorwärtserreichbarkeitsanalyse und die Rückwärtserreichbarkeitsanalyse. Die Vorwärtserreichbarkeitsanalyse wird dazu benutzt, alle von der Menge der Anfangszustände S_0 einer Kripke-Struktur erreichbaren Zustände zu berechnen. Dies erfolgt durch wiederholtes Berechnen von Mengen von Nachfolgerzuständen gemäß *Successors(Z)*=$\{s \mid s' \in Z \wedge R(s', s)\}$, bis zum Erreichen eines Fixpunktes an Zuständen. Bei der Rückwärtserreichbarkeitsanalyse werden alle Vorgängerzustände einer gegebenen Zustandsmenge Z, durch wiederholtes Berechnen von Vorgängerzuständen gemäß *Predecessors(Z)*=$\{s \mid s' \in Z \wedge R(s, s')\}$ berechnet. Für die in dieser Arbeit zur Erreichbarkeitsanalyse durchgeführten Verifikationsexperimente wurde die Vorwärtserreichbarkeitsanalyse verwendet. Die in dieser Arbeit für Bildberechnungen eingeführten Ansätze und Verbesserungen können aber nach entsprechender Anpassung für die Berechnung von Vorgänger- und Nachfolgerzuständen benutzt werden. Also auch für die Rückwärtserreichbarkeitsanalyse und für die CTL- oder LTL-Modellprüfung.

Ein Algorithmus zur Vorwärtserreichbarkeitsanalyse ist in Algorithmus 1 zu finden. In der in diesem enthaltenen *while*-Schleife (Zeile 3) werden so lange Nachfolgerzustände von bisher unexplorierten Zuständen (*ToExplore* bzw. *Unexplored*) ermittelt, bis alle von den Anfangszuständen erreichbaren Zustände gefunden wurden.

Die Vorwärtserreichbarkeitsanalyse kann benutzt werden, um eine der wichtigsten Arten von Systemeigenschaften bei der Modellprüfung, nämlich Invarianten, zu prüfen. Invarianten sind Eigenschaften, die jeder von den Anfangszuständen eines Systems erreichbare Zustand erfüllen muss. Eine Prüfung der Invarianten kann dabei direkt in den Prozess der Zustandsraumexploration integriert werden. Invarianten entsprechen temporallogischen Formeln der Form **AG** f, wobei f eine Eigenschaft ohne temporale Operatoren und Pfad-

[1] Dies ist die am häufigsten anzutreffende Definition von Fairness. Weitere finden sich in [90].

Algorithmus 1: Symbolische Vorwärtserreichbarkeitsanalyse ausgehend von den Startzuständen S_0 einer Kripke-Struktur.

```
1 Reached = S_0
2 ToExplore = S_0
3 while ToExplore ≠ ∅ do
4 |   ExploredStates = Successors(ToExplore)
5 |   Unexplored = ExploredStates \ Reached
6 |   ToExplore = Unexplored
7 |   Reached = Reached ∪ Unexplored
```

quantoren ist, die sich nur auf einen Zustand bezieht. Die klassischen symbolischen CTL-Modellprüfer verwenden zur Berechnung der Gültigkeit von CTL-Formeln wiederholte Vorgängerberechnungen. Für die Berechnung der Gültigkeit von Eigenschaften wie **AG** f oder **EF**f wird auch die Vorwärtserreichbarkeitsanalyse benutzt. Die CTL-Modellprüfung kann aber auch hauptsächlich durch Benutzung von Nachfolgerberechnungen durchgeführt werden [119]. In [118, 119] finden sich Ergebnisse von Verifikationsexperimenten, bei denen die Benutzung von Vorgänger- und Nachfolgerberechnungen zur Überprüfung von in der temporalen Logik CTL formulierten Eigenschaften verglichen wird. Die dort präsentierten Laufzeiten der Verifikationsexperimente zeigen, dass die hauptsächlich auf Nachfolgerberechnungen basierende CTL-Modellprüfung zu besseren Laufzeiten führen kann.

Im Gegensatz zur Vorwärtserreichbarkeitsanalyse, die in der Regel mit der Menge der Anfangszustände gestartet wird und bei der nur Nachfolgerberechnungen durchgeführt werden, wird die Rückwärtserreichbarkeitsanalyse mit einer Menge von fehlerhaften Zuständen gestartet. Ausgehend von diesen wird die Menge der Zustände berechnet, von denen diese fehlerhaften Zustände erreicht werden können. Ist ein Anfangszustand des Systems in dieser Menge enthalten, so gibt es einen Berechnungspfad des Systems, der zu einem fehlerhaften und damit unerwünschten Zustand führt. Mit Hilfe eines dafür von einem Modellprüfer ausgegebenen Gegenbeispiels kann das Erreichen des fehlerhaften Zustands analysiert und behoben werden.

2.1.7 Bekannte Modellprüfer

Im Folgenden werden einige bekannte Modellprüfer vorgestellt. Weitere Modellprüfer, die direkt mit Verfahren zur Ausnutzung von in einem System vorhandenen Symmetrien (siehe Abschnitt 2.4) entworfen wurden, werden in Abschnitt 2.4.3 beschrieben. Außerdem werden dort Erweiterungen von hier vorgestellten Modellprüfern um Symmetriereduktionstechniken erläutert. Nachfolgend werden in Abschnitt 2.1.7.1 zuerst Modellprüfer für die explizite Modellprüfung, bei denen eine explizite Repräsentation von Kripke-Strukturen verwendet wird und bei denen Zustände explizit abgespeichert werden, vorgestellt. Anschließend wird in Abschnitt 2.1.7.2 auf Modellprüfer, für die die symbolische Modellprüfung mit BDDs implementiert wurde, eingegangen. Diese verwenden eine symbolische

Repräsentation von Zustandsmengen und der Transitionsrelation durch BDDs und arbeiten mit Mengen von Zuständen und Transitionen.

2.1.7.1 Explizite Modellprüfer

Ein für die Verifikation von Protokollen entworfener expliziter Modellprüfer ist der Modellprüfer Murphi [63]. Seine Eingabesprache ermöglicht das Spezifizieren des Verhaltens eines Systems durch eine Menge von Regeln. Diese Regeln bestehen aus einer Bedingung und einer Aktion, die ausgeführt werden kann, wenn die Bedingung in einem Zustand erfüllt ist. Sind Bedingungen für die Ausführbarkeit mehrerer Regeln erfüllt, wird nichtdeterministisch eine Regel zur Ausführung ausgewählt und die angegebenen Anpassungen der Zustandsvariablen werden durchgeführt. Die Eingabesprache von Murphi ist damit gut geeignet, um nebenläufige Systeme zu beschreiben. Murphi kann Deadlocks detektieren, eingegebene Behauptungen (engl. asserts) und Invarianten überprüfen. Aus einer eingegebenen Verifikationsaufgabe erstellt Murphi ein ausführbares C++-Programm. Durch dessen Ausführung werden die spezifizierten Eigenschaften für das eingegebene Modell überprüft.

SPIN [111] ist ein weitverbreiteter expliziter Modellprüfer, der die Überprüfung von LTL-Eigenschaften (siehe Abschnitt 2.1.3.3) erlaubt und der für die Verifikation verteilter Algorithmen entworfen wurde. Als Eingabesprache verwendet er Promela [98], welches vor allem zur Beschreibung der Koordinations- und Synchronisationsaspekte in nebenläufigen Systemen geeignet ist. Durch Implementierung vieler Methoden, um den Zustandsraum klein zu halten (z.B. Benutzung von *on-the-fly Verifikationsalgorithmen* (siehe Abschnitt 2.1.4) oder Reduktion des Zustandsraums durch Halbordnung (engl. partial order reduction) [8, 56, 112, 157]), können mit ihm auch Systeme mit einigen Millionen erreichbaren Zuständen verifiziert werden [20].

Ein weiterer expliziter Modellprüfer ist der Modellprüfer PROB [134], der für die *B-Methode* (engl. B method) [3] entworfen wurde. Die B-Methode ist ein Prozess zur Entwicklung von Software, der bei abstrakten Spezifikationen des Verhaltens der Software beginnt und aus schrittweisen Verfeinerungen der Spezifikation bis zur konkreten Implementierbarkeit besteht. PROB kann dabei zum Beispiel zur Konsistenzprüfung von Spezifikationen mit gegebenen Invarianten verwendet werden.

2.1.7.2 Symbolische Modellprüfer

Ein erfolgreicher BDD-basierter symbolischer Modellprüfer ist SMV (Symbolic Model Verifier) [142]. Seine Eingabesprache erlaubt die modulare hierarchische Beschreibung eines Systems und die Mehrfachverwendung von spezifizierten Modulen. Er kann zur Modellprüfung asynchroner und synchroner Systeme benutzt werden. Die zu überprüfenden Eigenschaften können bei ihm mit der temporalen Logik CTL angegeben werden. Eine Erweiterung und Neuimplementierung des Modellprüfers SMV ist der Modellprüfer NuSMV (New Symbolic Model Verifier) [50]. Neben der bereits in SMV implementierten BDD-basierten symbolischen Modellprüfung erlaubt NuSMV auch die Benutzung von auf dem Erfüllbarkeitsproblem (SAT, engl. satisfiability) basierenden Techniken zur Modellprü-

fung. Durch deren Anwendung soll eine bei der BDD-basierten Modellprüfung mögliche Explosion des Speicherbedarfs vermieden werden [23]. Als Eingabe akzeptiert er Systeme, die in einer Erweiterung der Eingabesprache des Modellprüfers SMV beschrieben wurden. Die zu überprüfenden Eigenschaften können sowohl in der temporalen Logik CTL, als auch in der temporalen Logik LTL formuliert werden.

Ein in der Industrie entwickelter Modellprüfer, der ebenfalls auf dem Modellprüfer SMV basiert und bei dem besonders auf einfache Bedienbarkeit und praktische Einsetzbarkeit geachtet wurde, ist RuleBase [15, 16]. Er wurde für die Verifikation von Hardware entwickelt und besitzt als Eingabesprachen VHDL und Verilog. Zur Erleichterung der Eingabe von Eigenschaften wird eine eigene Sprache *Sugar*, die auf der temporalen Logik CTL aufsetzt, verwendet. Außerdem wurden für ihn viele Methoden zur Verkleinerung des Zustandsraums implementiert.

Der BDD-basierte Modellprüfer BOOM [12] erlaubt die Verifikation von nebenläufigen Booleschen Programmen. Er wurde zur Benutzung bei der Modellprüfung mit Prädikatenabstraktion (engl. predicate abstraction) [102] implementiert. Bei der Prädikatenabstraktion werden Programme höherer Programmiersprachen (z.B. C, C++) in Programme übersetzt, die nur Boolesche Variablen besitzen. Die Booleschen Variablen speichern Wahrheitswerte von Prädikaten über den Variablen des ursprünglichen Programms. Damit werden die Verhaltensweisen des ursprünglichen Programms überapproximiert.

2.2 Binäre Entscheidungsdiagramme (BDDs)

In 2.1.5 wurde gezeigt, wie Zustandsmengen und Mengen von Transitionen von Kripke-Strukturen durch Boolesche Funktionen, nämlich entsprechenden charakteristischen Funktionen, dargestellt werden können. Binäre Entscheidungsdiagramme (engl. Binary Decision Diagrams, BDDs) sind eine Datenstruktur zur Repräsentation von Booleschen Funktionen. Ein erfolgreicher Ansatz zur symbolischen Modellprüfung ist die Verwendung von BDDs als Datenstruktur zur Repräsentation von Mengen. BDDs basieren auf der Shannon-Expansion [176] und wurden in [4, 132] als allgemeine binäre Entscheidungsdiagramme eingeführt. Zu großer Bekanntheit, vor allem im Bereich der Verifikation, gelangten BDDs erst durch Bryant [33, 35]. Bryant führte Verfeinerungen ein, mit denen die kanonische Repräsentationen von Mengen durch BDDs möglich wurde. Die von Bryant verfeinerten BDDs werden reduzierte geordnete binäre Entscheidungsdiagramme (engl. Reduced Ordered Binary Decision Diagrams, ROBDDs) genannt. Sie liefern für viele praktisch relevante Boolesche Funktionen kompakte Darstellungen, zu deren Manipulation Bryant zusätzlich leistungsfähige Algorithmen entwickelt hat. Durch die Benutzung von ROBDDs entstanden effiziente Algorithmen für die symbolische Modellprüfung mit ROBDDs [38, 58, 159]. In den nachfolgenden Abschnitten werden Grundlagen zu ROBDDs eingeführt. Weitere Informationen finden sich z.B. in folgenden Büchern [8, 56, 73, 74, 144].

2.2.1 Reduzierte geordnete binäre Entscheidungsdiagramme (ROBDDs)

Geordnete binäre Entscheidungsdiagramme (engl. Ordered Binary Decision Diagrams, OBDDs) sind binäre Entscheidungsbäume, bei denen die Variablen entlang jeden Pfades von der Wurzel zu den Terminalknoten in der gleichen Reihenfolge vorkommen. Dazu wird eine feste Variablenordnung benutzt, die die Reihenfolge, in der die Variablen auf den Pfaden eines BDDs aufeinander folgen müssen, vorgibt.

Definition 2.16. (Variablenordnung) *Gegeben sei eine endliche Menge von Booleschen Variablen* $Var = \{z_1, z_2, ..., z_n\}$ *und eine Permutation* $level : \{1, 2, ..., n\} \to \{1, 2, ..., n\}$. *Mit level wird eine totale Ordnung* $<_{level}$ *auf der Menge der Variablen durch* $z_i <_{level} z_j \Leftrightarrow level(i) < level(j)$ *induziert. Die totale Ordnung* $<_{level}$ *wird Variablenordnung genannt und die Permutation level ordnet jeder Variablen* z_i *eine Position* $level(i) \in \{1, 2, ..., n\}$ *innerhalb der Variablenordnung zu.*

Ein OBDD ist ein binäres Entscheidungsdiagramm, das eine vorgegebene Variablenordnung benutzt.

Definition 2.17. (Geordnetes binäres Entscheidungsdiagramm (OBDD), [8]) *Gegeben sei eine endliche Menge* $Var = \{x_1, x_2, ..., x_n\}$ *an Booleschen Variablen und eine Variablenordnung* $<_{level}$. *Ein geordnetes binäres Entscheidungsdiagramm (OBDD) über der Variablenmenge* Var *mit Variablenordnung* $<_{level}$ *ist ein Tupel*
$\mathfrak{D} = (V, V_I, V_T, succ_0, succ_1, var, val, v_0)$ *mit folgenden Eigenschaften:*

- V *ist eine endliche Knotenmenge, die aus der Vereinigung der disjunkten Mengen der inneren Knoten* V_I *und der Terminalknoten* V_T *besteht.*
- $succ_0, succ_1 : V_I \to V$ *sind Nachfolgerfunktionen, die jedem inneren Knoten* $v \in V_I$ *ein* 0*-Kind,* $succ_0(v) \in V$, *und ein* 1*-Kind,* $succ_1(v) \in V$, *zuordnen.*
- $var : V_I \to Var$ *ist eine Markierungsfunktion, die jedem inneren Knoten* v *eine Variable* $var(v) \in Var$ *zuordnet.*
- $val : V_T \to \{0, 1\}$ *ist eine Markierungsfunktion, die jedem Terminalknoten den Funktionswert* 0 *oder* 1 *zuordnet.*
- $v_0 \in V$ *ist ein ausgezeichneter Wurzelknoten, der als einziger Knoten aus* V *keine eingehenden Kanten besitzt.*
- *Auf jedem Pfad vom Wurzelknoten zu einem Terminalknoten kommt jede Variable höchstens einmal vor und für jeden inneren Knoten* $v \in V_I$ *gilt:*
  - $var(v) <_{level} var(succ_0(v))$, *falls* $succ_0(v)$ *ebenfalls ein innerer Knoten ist.*
  - $var(v) <_{level} var(succ_1(v))$, *falls* $succ_1(v)$ *ebenfalls ein innerer Knoten ist.*

Mit OBDDs können Boolesche Funktionen repräsentiert werden, indem jedem Knoten $v \in V$ eines OBDDs eine Boolesche Funktion f_v zugeordnet wird, die der Knoten repräsentiert. OBDDs basieren auf der Shannon-Zerlegung, die eine Zerlegung einer Booleschen

Funktion und ihre Darstellung durch einfachere Kofaktoren beschreibt. Jeder Knoten eines OBDDs stellt eine Aufspaltung der von ihm repräsentierten Booleschen Funktion in Kofaktoren bezüglich der dem Knoten zugeordneten Booleschen Variablen dar.

Definition 2.18. (Kofaktor, [73]) *Sei $f : \mathbb{B}^n \to \mathbb{B}$ eine Boolesche Funktion über der Variablenmenge $Var = \{x_1, x_2, ..., x_n\}$, x_i eine Boolesche Variable und $c \in \mathbb{B}$. Dann ist der Kofaktor von f nach $x_i = c$ definiert durch*

$$f|_{x_i=c} = f(x_1, x_2, ..., x_{i-1}, c, x_{i+1}, ..., x_n). \tag{2.3}$$

Satz 2.19. (Shannon-Zerlegung, [73, 176]) *Sei $f : \mathbb{B}^n \to \mathbb{B}$ eine Boolesche Funktion über der Variablenmenge $Var = \{x_1, x_2, ..., x_n\}$ und x_i eine Boolesche Variable, dann gilt:*

$$f = \overline{x_i} \cdot f|_{x_i=0} \vee x_i \cdot f|_{x_i=1} \tag{2.4}$$

Durch mit dem Wurzelknoten eines OBDDs beginnende rekursive Anwendung der Shannon-Zerlegung, kann für jeden Knoten eines OBDDs die von ihm repräsentierte Boolesche Funktion ermittelt werden.

Definition 2.20. (Semantik von OBDDs, [152, 175]) *Gegeben sei ein OBDD $\mathfrak{D} = (V, V_I, V_T, succ_0, succ_1, var, val, v_0)$ über der Variablenmenge $Var = \{x_1, x_2, ..., x_n\}$ mit Variablenordnung $<_{level}$. Sei $v \in V$ ein Knoten von $\mathfrak{D}$, dann ist die durch v definierte Boolesche Funktion f_v*

- *die Konstante 0, mit $f_v(x_1, x_2, ..., x_n) = 0 \quad \forall (x_1, x_2, ..., x_n) \in \{0,1\}^n$, falls $v \in V_T$ und $val(v) = 0$.*
- *die Konstante 1, mit $f_v(x_1, x_2, ..., x_n) = 1 \quad \forall (x_1, x_2, ..., x_n) \in \{0,1\}^n$, falls $v \in V_T$ und $val(v) = 1$.*
- *definiert durch $f_v(x_1, x_2, ..., x_n) = \overline{x_i} \cdot f_{succ_0(v)}(x_1, x_2, ..., x_n) \vee x_i \cdot f_{succ_1(v)}(x_1, x_2, ..., x_n)$ $\forall (x_1, x_2, ..., x_n) \in \{0,1\}^n$, falls $v \in V_I$ und $var(v) = x_i$.*

Für die Boolesche Funktion eines OBDDs $\mathfrak{D}$ gilt $f_{\mathfrak{D}} = f_{v_0}$.

Den Funktionswert eines OBDDs für eine Boolesche Funktion über einer Menge von Booleschen Variablen $Var = \{x_1, x_2, ..., x_n\}$ und einer Belegung dieser Variablen erhält man durch Traversieren des OBDDs vom Wurzelknoten $v_0 \in V$ bis zu einem Terminalknoten $v_T \in V_T$. Der Funktionswert $val(v_T)$ des Terminalknotens gibt dabei den Wert der Booleschen Funktion für die verwendete Belegung der Variablen an. Neben der Benutzung einer festen Variablenordnung für binäre Entscheidungsdiagramme führte Bryant Reduktionsregeln ein, mit denen redundante Knoten und mehrfach vorkommende isomorphe Teilgraphen eliminiert werden können.

Definition 2.21. (Isomorphe binäre Entscheidungsdiagramme, [152]) *Zwei geordnete binäre Entscheidungsdiagramme $\mathfrak{D}_A = (V_A, V_{I_A}, V_{T_A}, succ_{0_A}, succ_{1_A}, var_A, val_A, v_{0_A})$ und $\mathfrak{D}_B = (V_B, V_{I_B}, V_{T_B}, succ_{0_B}, succ_{1_B}, var_B, val_B, v_{0_B})$ über einer Menge von Booleschen Variablen $Var = \{x_1, x_2, ..., x_n\}$ heißen isomorph, wenn es eine bijektive Abbildung $\gamma : V_A \to V_B$ gibt, so dass für jeden inneren Knoten $v_{I_A} \in V_{I_A}$ gilt*

- $\gamma(v_{I_A})$ *ist auch in* $\mathfrak{D}_B$ *ein innerer Knoten.*
- $var_A(v_{I_A}) = var_B(\gamma(v_{I_A}))$.
- $\gamma(succ_{0_A}(v_{I_A})) = (succ_{0_B}(\gamma(v_{I_A})))$ *und* $\gamma(succ_{1_A}(v_{I_A})) = (succ_{1_B}(\gamma(v_{I_A})))$.

und für jeden Terminalknoten $v_{T_A} \in V_{T_A}$ *gilt*

- $\gamma(v_{T_A})$ *ist auch in* $\mathfrak{D}_B$ *ein Terminalknoten.*
- $val_A(v_{T_A}) = val_B(\gamma(v_{T_A}))$.

Durch das Eliminieren redundanter Knoten und das Eliminieren von in einem OBDD enthaltenen isomorphen Teilgraphen erhält man ein reduziertes geordnetes binäres Entscheidungsdiagramm (ROBDD).

Definition 2.22. (Reduziertes geordnetes binäres Entscheidungsdiagramm (ROBDD), [152, 175]) *Ein OBDD* $\mathfrak{D} = (V, V_I, V_T, succ_0, succ_1, var, val, v_0)$ *heißt genau dann reduziert, wenn*

- *es keinen Knoten* $v \in V_I$ *mit* $succ_0(v) = succ_1(v)$ *gibt.*
- *es keine zwei Knoten* $v, t \in V$ *gibt, so dass die Teildiagramme mit den Wurzeln* v *und* t*, die aus allen Knoten und Kanten bestehen, die von* v *bzw. von* t *erreicht werden können, isomorph sind.*

ROBDDs sind eine kanonische Darstellung Boolescher Funktionen [33].

Satz 2.23. (Kanonizität von ROBDDs, [152]) *Gegeben sei eine endliche Menge* $Var = \{x_1, x_2, ..., x_n\}$ *von booleschen Variablen und eine Variablenordnung* $<_{level}$*. Dann gibt es zu jeder Booleschen Funktion* $f : \mathbb{B}^n \to \mathbb{B}$ *über der Variablenmenge* Var *(bis auf Isomorphie) genau einen ROBDD mit Variablenordnung* $<_{level}$*, der* f *beschreibt.*

Im Vergleich mit OBDDs können ROBDDs eine deutlich geringere Knotenanzahl besitzen. Ein Beispiel eines OBDDs und eines ROBDDs zur Booleschen Funktion $f = x_1x_2 \vee x_3x_4$ und der Variablenordnung $<_{level_1}$, mit $x_1 <_{level_1} x_2 <_{level_1} x_3 <_{level_1} x_4$, ist in Abbildung 2.6 angegeben. Sind in der vorliegenden Arbeit Beispiele für BDDs in Grafiken angegeben, so zeigen gestrichelte Pfeile immer auf das 0-Kind eines Knotens und durchgehende Pfeile auf das 1-Kind eines Knotens.

Im Rest der vorliegenden Arbeit werden zur Repräsentation von Booleschen Funktionen, wenn von BDDs gesprochen wird und keine explizite Kennzeichnung als OBDD erfolgt, immer ROBDDs verwendet. Der Grund dafür ist, dass ROBDDs die für die Modellprüfung wichtige Eigenschaft der Kanonizität aufweisen. Außerdem sind sie aufgrund der Reduktionsregeln kompakter als OBDDs. In Definition 2.24 wird nachfolgend der Begriff der wesentlichen Variable eingeführt.

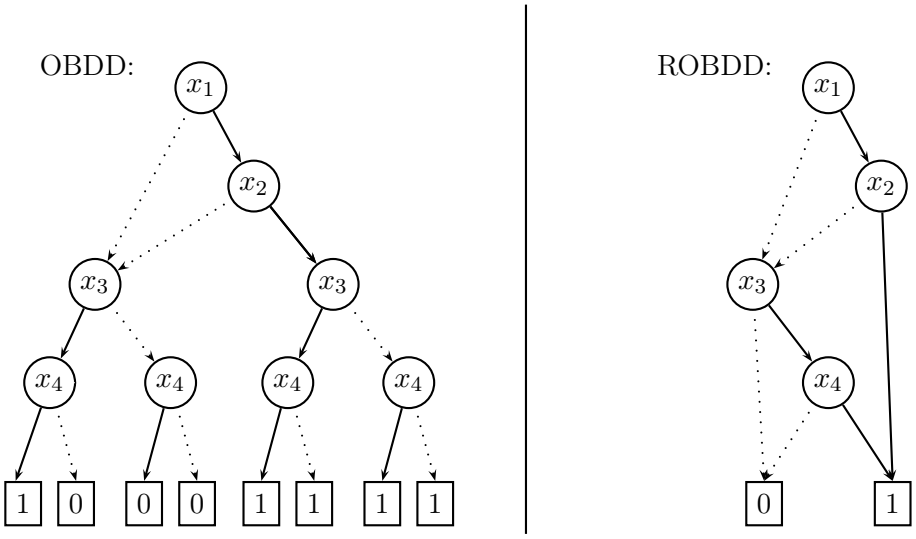

Abbildung 2.6: Repräsentation der Booleschen Funktion $f(x_1, x_2, x_3, x_4) = x_1x_2 \vee x_3x_4$ durch einen OBDD und einen ROBDD mit der Variablenordnung $<_{level_1}$, mit $x_1 <_{level_1} x_2 <_{level_1} x_3 <_{level_1} x_4$.

Definition 2.24. (Wesentliche Variablen, [144]) *Sei $f : \mathbb{B}^n \to \mathbb{B}$ eine Boolesche Funktion über der Variablenmenge $Var = \{x_1, x_2, ..., x_n\}$ und x_i eine Boolesche Variable aus dieser Variablenmenge. Dann heißt die Variable x_i wesentlich für f, wenn es eine Eingabebelegung $(x_1, x_2, ..., x_n) \in \{0,1\}^n$ gibt mit*

$$f|_{x_i=1} \neq f|_{x_i=0}. \tag{2.5}$$

Der aus einer Eingabebelegung resultierende Funktionswert einer Booleschen Funktion ist nur von den den wesentlichen Variablen zugeordneten Werten in einer Eingabebelegung abhängig. Er ist unabhängig von den Booleschen Werten nicht wesentlicher Variablen, da sich für beide Belegungen solcher Variablen der gleiche Funktionswert ergibt. Durch den für beide Belegungen nicht wesentlicher Variablen resultierenden gleichen Funktionswert, sind die beiden Nachfolgerknoten von in OBDDs für solche Variablen vorhandenen Knoten v gleich ($succ_0(v) = succ_1(v)$). Der Grund ist, dass beide Variablenbelegungen für diese Variablen zur gleichen mit den Nachfolgerknoten im OBDD zu repräsentierenden Funktion führen. Nach Definition 2.22 sind in ROBDDs keine Knoten vorhanden, bei denen die beiden Nachfolgerknoten gleich sind. Daher kommen in ROBDDs keine Knoten für nicht wesentliche Variablen vor.

Mit Hilfe des Begriffs der wesentlichen Variablen können Funktionen verschiedener Stelligkeit in Verbindung gebracht werden [144]. Eine Funktion $f : \mathbb{B}^n \to \mathbb{B}$ über der Variablenmenge $Var_1 = \{x_1, x_2, ..., x_n\}$ kann durch Hinzufügen einer nicht wesentlichen Variablen x_{n+1} als Funktion $f : \mathbb{B}^{n+1} \to \mathbb{B}$ über der Variablenmenge $Var_2 = \{x_1, x_2, ..., x_n, x_{n+1}\}$

betrachtet werden. Die neue Variable hat keinen Einfluss auf den Funktionswert und dient lediglich der Angleichung der Definitionsbereiche. Damit können Funktionen für Kofaktoren einer Booleschen Funktion über der ursprünglichen Variablenmenge der Booleschen Funktion, oder über den Variablen, denen durch die Kofaktoroperation kein fester Wert zugewiesen wurde, angegeben werden. Die Beweise zur Korrektheit der in den Kapiteln 3, 4, 5 und 6 eingeführten neuen Algorithmen werden immer mit allen zur Kodierung einer Kripke-Struktur nötigen Booleschen Variablen geführt. Bei Betrachtung von Kofaktoren über den wesentlichen Variablen, können diese durch Hinzufügen von unwesentlichen Variablen auf die in den Sätzen angegebene Formulierung erweitert werden.

2.2.2 Variablenordnungen und dynamisches Umordnen von Variablen

Eine wichtige Voraussetzung dafür, dass ROBDDs kanonische Repräsentanten für Boolesche Funktionen sind, ist das Verwenden einer festen Variablenordnung. Von der benutzten Variablenordnung hängt ebenfalls die Größe von ROBDDs sehr stark ab. Es gibt Boolesche Funktionen, für die ROBDDs bei jeder Variablenordnung exponentielle Größe besitzen, wie zum Beispiel die Multiplikation [34]. Außerdem gibt es Boolesche Funktionen, für die für verschiedene Variablenordnungen erzeugte ROBDDs lineare bis exponentielle Größe in der Anzahl der Variablen besitzen. Ein solches Beispiel sind die Booleschen Funktionen $f_n(x_1, x_2, ..., x_{2n-1}, x_{2n}) = x_1x_2 \vee x_3x_4 \vee ... \vee x_{2n-1}x_{2n}$ für $n \geq 1$ [84]. Für $n = 2$ sind ROBDDs der korrespondierenden Booleschen Funktion in den Abbildungen 2.6 und 2.7 zu sehen. Abbildung 2.6 zeigt rechts ein ROBDD für die Boolesche Funktion $f(x_1, x_2, x_3, x_4) = x_1x_2 \vee x_3x_4$ mit der Variablenordnung $<_{level_1}$, mit $x_1 <_{level_1} x_2 <_{level_1} x_3 <_{level_1} x_4$. In Abbildung 2.7 ist ein ROBDD für diese Boolesche Funktion mit der Variablenordnung $<_{level_2}$, mit $x_1 <_{level_2} x_3 <_{level_2} x_2 <_{level_2} x_4$, dargestellt. Wie an den Abbildungen zu sehen ist, wird bei Verwendung der Variablenordnung $<_{level_2}$ eine größere Knotenanzahl zur Repräsentation der Booleschen Funktion benötigt. In [84] wurde gezeigt, dass ROBDDs für die oben beschriebenen Booleschen Funktionen f_n mit der Variablenordnung $<_{level}$, mit $x_1 <_{level} x_3 <_{level} ... <_{level} x_{2n-1} <_{level} x_2 <_{level} x_4 <_{level} ... <_{level} x_{2n}$, exponentielle Größe besitzen.

Da die Größe von ROBDDs für eine Boolesche Funktion in Abhängigkeit der gewählten Variablenordnung sehr stark variieren kann, ist es wichtig eine Variablenordnung zu verwenden, bei der eine geringe Knotenanzahl benötigt wird. Neben eines geringen Speicherbedarfs führt dies dazu, dass Operationen mit ROBDDs effizienter durchgeführt werden können (siehe Abschnitt 2.2.4). Allerdings ist das Problem für eine gegebene Boolesche Funktion eine Variablenordnung mit minimaler Anzahl an ROBDD-Knoten zu finden NP-vollständig [28].

Um in der Praxis dennoch automatisch gute Variablenordnungen ermitteln zu können, die ROBDDs mit möglichst kleiner Knotenzahl liefern, wurden Heuristiken entwickelt. Diese kann man in zwei Klassen einteilen, nämlich in statische und in dynamische Verfahren. Statische Verfahren ermitteln vor der Erzeugung eines ROBDDs eine möglichst günstige Variablenordnung. Dazu analysieren sie die Struktur einer gegebenen Spezifikation und leiten daraus eine Variablenordnung ab. Häufig lassen sich damit gute Variablenordnungen direkt ableiten, z.B. bei Schaltkreisen durch Analyse der Topologie. Beispiele für

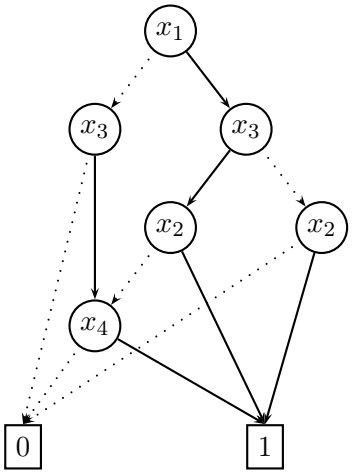

Abbildung 2.7: Repräsentation der Booleschen Funktion $f(x_1, x_2, x_3, x_4) = x_1x_2 \vee x_3x_4$ durch einen ROBDD mit der Variablenordnung $<_{level_2}$, mit $x_1 <_{level_2} x_3 <_{level_2} x_2 <_{level_2} x_4$.

statische Verfahren finden sich in [91, 92, 93, 94, 140, 148]. Statische Verfahren haben aber einige Nachteile. Sie lassen sich oft nur problemspezifisch anwenden. Außerdem bleibt die gewählte Variablenordnung während der Benutzung von ROBDDs für eine Anwendung gleich. Bei vielen Anwendungen, wie z.B. bei der Modellprüfung, kann sich die optimale Variablenordnung aber während der Ausführung ändern. Eine zu Beginn gewählte gute Variablenordnung kann dann zu unnötig hohem Speicherbedarf führen.

Deshalb wurden dynamische Verfahren entwickelt, die die Variablenordnung während der Abarbeitung von Aufgaben dynamisch verändern können. Bei diesen kann die Variablenordnung von ROBDDs während oder nach deren Aufbau und auch mehrfach zu verschiedenen Zeitpunkten geändert werden. Damit unnötiger Aufwand durch das Umordnen von Variablen vermieden wird, ist es dabei häufig sinnvoll, die Suche nach einer anderen besseren Variablenordnung erst anzustoßen, wenn bestimmte Voraussetzungen erfüllt sind. Dies kann zum Beispiel das Überschreiten einer bestimmten zur Repräsentation von verwendeten BDDs benötigten Knotenanzahl sein. Eine Basisoperation bei den dynamischen Verfahren ist das Umordnen von zwei in einer gegebenen Variablenordnung benachbarten Booleschen Variablen x_i und x_{i+1} [94, 120]. Dies ist effizient, alleine durch Umordnen von ROBDD-Knoten für die benachbarten Variablen x_i und x_{i+1}, möglich. Die anderen Knoten des ROBDDs ändern sich nicht.

Ein sehr häufig eingesetztes dynamisches Verfahren zum Finden guter Variablenordnungen ist der Sifting-Algorithmus [172]. Bei gegebener Boolescher Funktion über einer Variablenmenge $Var = \{x_1, x_2, ..., x_n\}$, wird beim Sifting-Algorithmus für jede Variable $x_i \in \{x_1, x_2, ..., x_n\}$ die optimale Position in der aktuellen Variablenordnung gesucht. Dies wird dadurch erreicht, dass die Variable x_i in der aktuellen Variablenordnung durch Vertauschung mit benachbarten Variablen an alle anderen $n-1$ Stellen der Variablenordnung

geschoben wird. Die Variablenordnung zwischen den anderen Variablen wird dadurch nicht verändert. Anschließend wird für jede Variable x_i die Variablenordnung ausgewählt, die zur minimalen Anzahl an ROBDD-Knoten führt. Das Verfahren terminiert nach $\mathcal{O}(n^2)$ Vertauschungen von Variablen. Obwohl der Sifting-Algorithmus häufig gute Variablenordnungen liefert, besitzt er auch Schwächen. Ein Problem des Algorithmus ist, dass er zum Finden einer möglichst optimalen Variablenordnung für jede Variable die absolut beste Position in einer gegebenen Variablenordnung sucht und dabei relative Positionen von Variablen zueinander nicht direkt berücksichtigt [155]. Dies verursacht Probleme wenn sich Variablen gegenseitig anziehen und in guten Variablenordnungen nahe beieinander angeordnet sein müssen. Dann können durch den Sifting-Algorithmus Gruppen von sich gegenseitig anziehenden Variablen nahe beieinander in suboptimalen Positionen in der Variablenordnung platziert werden. Ein Beispiel dafür sind zwei benachbarte Variablen, die sich gegenseitig stark anziehen. Verschiebt der Sifting-Algorithmus eine der beiden Variablen, dann wird diese Variable aufgrund der starken Anziehungskräfte mit großer Wahrscheinlichkeit wieder in der Nähe der benachbarten Variablen platziert. Die optimale Position der beiden Variablen kann aber weit entfernt von der aktuellen Position in der Variablenordnung sein. Zur Behebung dieses Problems wurden Sifting-Algorithmen entwickelt, die anstatt einzelner Variablen Gruppen von Variablen auf einmal verschieben (siehe z.B. [156, 155]).

Das Bestimmen von Gruppen von zusammenhängenden und zusammen zu verschiebenden Variablen kann auf zwei Arten möglich sein. Gruppen von Variablen können automatisch während dem Sifting-Algorithmus bestimmt werden, oder vom Benutzer aufgrund von Wissen über die Anwendung festgelegt werden. Bei den für die vorliegende Arbeit mit dynamischem Umordnen der Variablenordnung durchgeführten Verifikationsexperimenten, wurden immer vor Beginn der Verifikationsexperimente festgelegte Gruppierungen von Variablen benutzt (siehe Abschnitt 8.3). Der für die Verifikationsexperimente verwendete Modellprüfer Sviss (siehe Abschnitt 2.4.3.3 und [191, 190]) benutzt für ROBDDs und für Operationen mit ROBDDs das ROBDD-Programmpaket CUDD [181]. In diesem sind mehrere Verfahren zum dynamischen Umordnen der Variablenordnung implementiert. Bei den Verifikationsexperimenten mit dynamischem Umordnen von Variablen wurde davon der normale Sifting-Algorithmus [172] zum Umordnen einzelner Variablen innerhalb von Gruppen von Variablen verwendet. Zum Umordnen der zu Beginn der Verifikationsläufe fest gebildeten Gruppen von Variablen wurde der Algorithmus aus [155] benutzt. Dieser ist eine Erweiterung des Sifting-Algorithmus, der es erlaubt benachbarte Variablen zu Gruppen von Variablen zusammen zu fassen, deren Position in der Variablenordnung dann gemeinsam und zusammenhängend verändert wird. Neben dem Sifting-Algorithmus gibt es noch weitere dynamische Verfahren zum Finden günstiger Variablenordnungen. Artikel über weitere Verfahren sind z.B. [27, 72, 85, 117].

2.2.3 Implementierung von ROBDD-Paketen

Um ROBDDs und effiziente Implementierungen von Operationen mit ROBDDs anwendungsunabhängig zur Verfügung zu stellen, wurden Programmpakete für ROBDDs entwickelt. Da die Leistungsfähigkeit von auf ROBDDs basierten Anwendungen stark von

der Effizienz von ROBDD-Operationen abhängt, wurde versucht diese Pakete möglichst schnell und speichereffizient zu implementieren. In diesem Abschnitt werden Konzepte vorgestellt, auf denen die meisten verfügbaren ROBDD-Programmpakete basieren.

Verfügbare ROBDD-Pakete sind z.B. CUDD [181], das Paket von Brace, Rudell und Bryant [31], das PUMA Paket [2], das CAL Paket [173] und das ABCD Paket [22]. Der für die Verifikationsexperimente in dieser Arbeit verwendete Modellprüfer Sviss (siehe Abschnitt 2.4.3.3) benutzt das ROBDD-Paket CUDD. Die Datenstruktur zur Repräsentation von ROBDD-Knoten $v \in V$ (siehe Definition 2.17 (OBDD) in Abschnitt 2.2.1) besitzt als Grundelemente einen Variablenindex und zwei Zeiger auf Nachfolgerknoten, die bei Nichtterminalknoten $v \in V_I$ auf das $0-Kind\,(succ_0(v))$ bzw. $1-Kind\,(succ_1(v))$ eines Knotens zeigen. Für Terminalknoten wird der Funktionswert häufig anstatt der Zeiger auf Nachfolgerknoten abgespeichert. Der Variablenindex entspricht dem Index der Variablen, für die ein Knoten gebaut wurde. In vielen Anwendungen, wie z.B. der Modellprüfung, müssen mehrere Boolesche Funktionen gleichzeitig durch ROBDDs dargestellt werden. Bei der Modellprüfung sind dies zum Beispiel ROBDDs zur Repräsentation der Transitionsrelationen und von Zustandsmengen. Da diese ROBDDs häufig nicht unabhängig voneinander sind, sondern oft gleiche Teile besitzen die in mehreren ROBDDs vorkommen, verwenden die meisten ROBDD-Programmpakete (wie auch CUDD) aus Effizienzgründen sogenannte SROBDDs (engl. Shared Reduced Ordered Binary Decision Diagrams, SROBDDs) [31, 148] zur Repräsentation Boolescher Funktionen. SROBDDs sind ROBDDs mit mehreren Wurzelknoten, die benutzt werden können, um mehrere ROBDDs mit gleicher Variablenordnung gemeinsam abzuspeichern. Dadurch müssen gleiche Teile der in einem SROBDD repräsentierten ROBDDs nur noch einmal gespeichert werden und der Speicherbedarf wird reduziert.

Bei der Verwendung von OBDDs ist es wünschenswert alle auftretenden OBDDs zu jedem Zeitpunkt reduziert als ROBDDs zu speichern. Alle durch Knoten repräsentierten Booleschen Funktionen, bzw. Unterfunktionen, sollen nur einmal vorhanden sein. Um dies zu erreichen, wird vor der Generierung eines neuen Knotens geprüft, ob schon ein Knoten existiert, der die gleiche Boolesche Funktion repräsentiert. Damit dies effizient möglich ist verwenden die meisten ROBDD-Pakete eine Eindeutigkeitstabelle (engl. Unique Table). Die Eindeutigkeitstabelle wird benutzt, um ROBDDs jederzeit reduziert zu halten. Sie wird üblicherweise als Hash-Tabelle realisiert und speichert relevante Informationen über bereits existierende Knoten. Bevor ein neuer ROBDD-Knoten erzeugt wird, wird geschaut ob der Knoten in der Eindeutigkeitstabelle bereits existiert. Nur falls der Knoten noch nicht existiert, wird er neu erzeugt. Ansonsten wird der bereits existierende Knoten ermittelt. Um zeitaufwendige mehrfache Berechnungen gleicher Ergebnisse bzw. Zwischenergebnisse von Operationen (z.B. UND-Verknüpfung mit gleichen ROBDDs als Operanden) zu vermeiden, wird häufig eine Ergebnistabelle (engl. Computed Table) verwendet. In der Ergebnistabelle werden bereits berechnete Zwischenergebnisse von Operationen mit ROBDDs gespeichert. Wenn Zwischenergebnisse bereits in der Ergebnistabelle gespeichert sind, können sie bei erneuter Anfrage schnell zur Verfügung gestellt werden. Sie wird üblicherweise ebenfalls als Hash-Tabelle realisiert. Da Ergebnisse mehrerer möglicherweise verschiedener Operationen in der Ergebnistabelle gespeichert werden können,

wird in den Einträgen der Ergebnistabelle neben den Operanden und dem Ergebnis einer Operation noch der Typ der durchgeführten Operation gespeichert.

Zur Durchführung der Negation von durch ROBDDs repräsentierten Booleschen Funktionen in konstanter Zeit und zur Reduktion der Größe von ROBDDs, wurde die Verwendung von Komplementkanten vorgeschlagen [126, 139]. Ohne Komplementkanten könnte die Generierung der Negation $\overline{f}$ einer durch einen ROBDD repräsentierten Booleschen Funktion f durch Vertauschung der Werte der beiden Terminalknoten des ROBDDs durchgeführt werden. Da die meisten ROBDD-Pakete ROBDDs aber in Form von SROBDDs abspeichern, ist es nicht möglich, einfach Knoten eines ROBDDs abzuändern. Dadurch könnten andere ROBDDs beeinflusst und verändert werden. Deshalb müsste ein neuer ROBDD für die negierte Boolesche Funktion aufgebaut werden, was Rechenzeit und zusätzlichen Speicherplatz erfordern würde. Komplementkanten können hingegen durch Verwendung eines Komplementbits, das in einem freien Bit von Zeigern von ROBDD-Knoten auf Nachfolgerknoten untergebracht wird, ohne zusätzlichen Speicheraufwand implementiert werden. Wenn das Komplementbit in einem Zeiger nicht gesetzt ist, wird die Funktion des ROBDDs, auf den der Zeiger zeigt, wie gewöhnlich interpretiert. Bei gesetztem Komplementbit wird statt der Booleschen Funktion f, die durch den mit dem Zeiger verbundenen ROBDD repräsentiert wird, der nachfolgende ROBDD als Repräsentation der Funktion $\overline{f}$ interpretiert. Damit können die Booleschen Funktionen f und $\overline{f}$ durch den gleichen ROBDD repräsentiert werden, nur zeigt im Fall von $\overline{f}$ ein Zeiger mit gesetztem Komplementbit auf die Wurzel des ROBDDs. Die Kanonizität kann durch Beschränkung der Zeiger auf denen Komplementbits gesetzt werden dürfen erreicht werden [144].

Neben diesen Komponenten besitzen ROBDD-Pakete häufig noch Implementierungen verschiedener Algorithmen zur Bestimmung günstiger Variablenordnungen (siehe Abschnitt 2.2.2). Außerdem besitzen sie effiziente Strategien zur Speicherverwaltung (sogenannte *Garbage Collection*-Strategies). Mit diesen wird entschieden, ob es sinnvoll ist den Speicherplatz nicht mehr benötigter Knoten frei zu geben, oder ob diese Knoten für den Fall, dass sie vielleicht bald wieder benötigt werden, noch nicht gelöscht werden.

2.2.4 Algorithmen für ROBDDs

In diesem Abschnitt werden für die Modellprüfung wichtige Algorithmen für Operationen mit ROBDDs vorgestellt. Diese sind auch in gängigen ROBDD-Paketen, wie z.B. CUDD [181], implementiert. Die effiziente Implementierung von ROBDD-Algorithmen ist wichtig, um bei der Benutzung von ROBDDs in Anwendungen eine möglichst geringe Laufzeit und einen möglichst geringen Speicherbedarf zu erhalten. Wie im vorangehenden Abschnitt 2.2.3 beschrieben, verwenden die meisten ROBDD-Pakete (z.B. CUDD) sogenannte SROBDDs (engl. Shared Reduced Ordered Binary Decision Diagrams) [31, 148] zur gleichzeitigen Speicherung mehrerer ROBDDs. SROBDDs sind ROBDDs mit mehreren Wurzelknoten, die es ermöglichen, dass gleiche Teile verschiedener ROBDDs nur einmal abgespeichert werden müssen. Bei Operationen mit einem ROBDD eines SROBDDs muss darauf geachtet werden, dass die anderen im SROBDD abgespeicherten ROBDDs nicht verändert werden. Zusätzlich muss das Ergebnis der gewünschten Operation korrekt berechnet werden. Da die meisten ROBDD-Pakete und das bei den Verifikationsexpe-

rimenten dieser Arbeit benutzte ROBDD-Paket CUDD SROBDDs zum Speichern von ROBDDs benutzen, wird in diesem Abschnitt immer von in einem SROBDD gespeicherten ROBDDs ausgegangen.

Zur effizienten Unterstützung von Operationen mit ROBDDs verwenden ROBDD-Pakete üblicherweise eine Eindeutigkeitstabelle (engl. Unique Table) und eine Ergebnistabelle (engl. Computed Table) (siehe Abschnitt 2.2.3). Im Folgenden wird in diesem Abschnitt die Eindeutigkeitstabelle als *UniqueTable* und die Ergebnistabelle als *ComputedTable* bezeichnet. Die UniqueTable wird benutzt, um ROBDDs jederzeit reduziert zu halten. In der ComputedTable werden bereits berechnete Zwischenergebnisse von Operationen mit ROBDDs gespeichert. Sie dient zur Vermeidung von Mehrfachberechnungen von Zwischenergebnissen. Zur Durchführung von Booleschen Operationen mit ROBDDs wurden effiziente Algorithmen entwickelt. Die Grundidee für diese Algorithmen basiert auf der Distributivität der Kofaktor-Berechnung bezüglich Boolescher Operationen.

Satz 2.25. (**Distributivität Kofaktor-Berechnung, [32]**) *Gegeben seien zwei Boolesche Funktionen f, g über einer Variablenmenge $Var = \{x_1, x_2, ..., x_n\}$, eine Boolesche Variable x_i, ein Wert $c \in \mathbb{B}$ und ein beliebiger zweistelliger Boolescher Operator $\circ$. Dann gilt:*

- $(f \circ g)|_{x_i=c} = f|_{x_i=c} \circ g|_{x_i=c}$.
- $\overline{f|_{x_i=c}} = \overline{f}|_{x_i=c}$.

Aus der Shannon-Zerlegung (siehe Abschnitt 2.2.1) und Satz 2.25 ergibt sich für Boolesche Funktionen f, f_1, f_2 und eine zweistellige Boolesche Operation $\circ$ direkt ein rekursives Berechnungsschema zur Berechnung von $f = f_1 \circ f_2$:

$$\begin{aligned} f &= \overline{x_i} \cdot f|_{x_i=0} \vee x_i \cdot f|_{x_i=1} \\ &= \overline{x_i} \cdot (f_1 \circ f_2)|_{x_i=0} \vee x_i \cdot (f_1 \circ f_2)|_{x_i=1} \\ &= \overline{x_i} \cdot (f_1|_{x_i=0} \circ f_2|_{x_i=0}) \vee x_i \cdot (f_1|_{x_i=1} \circ f_2|_{x_i=1}) \end{aligned} \tag{2.6}$$

Damit kann die Berechnung der Verknüpfung von Booleschen Funktionen durch zweistellige Boolesche Operatoren rekursiv durch Berechnung der Operationen für einfachere Teilprobleme, nämlich einfacheren Kofaktoren, durchgeführt werden. Die Teilergebnisse müssen danach jeweils nur noch zusammengesetzt werden. Dieses rekursive Berechnungsprinzip wird im Algorithmus für zweistellige Boolesche Operationen mit ROBDDs, der im Folgenden vorgestellt wird, verwendet. In diesem werden gemäß der für die ROBDDs verwendeten Variablenordnung rekursiv Kofaktoren bezüglich der momentan obersten Variablen, für die in einem der beteiligten (Teil-)ROBDDs für f_1 und f_2 ein Knoten vorkommt, berechnet. Da Nichtterminalknoten von ROBDDs eine Aufspaltung einer Booleschen Funktion gemäß der Shannon-Zerlegung repräsentieren, können solche Kofaktoren in konstanter Zeit berechnet werden. Für ROBDDs zu Booleschen Funktionen f_1, f_2 über einer Menge an Booleschen Variablen $Var = \{x_1, x_2, ..., x_n\}$ lassen sich für die Berechnung von Kofaktoren bezüglich der obersten Variable x_i, für die ein Knoten in einem der beiden ROBDDs vorhanden ist, zwei Fälle unterscheiden:

1. Falls die repräsentierte Boolesche Funktion f_1 nicht von der Variablen x_i abhängt, dass heißt es gilt $f_1 = f_1|_{x_i=0} = f_1|_{x_i=1}$, ist der resultierende Kofaktor die Funktion f_1 selbst und der ROBDD für den Kofaktor ist der ROBDD für die Funktion f_1. Das gleiche gilt entsprechend für f_2.

2. Falls die Boolesche Funktion f_1 von der Variablen x_i abhängt, entspricht
   - der Kofaktor $f_1|_{x_i=0}$ dem 0-Kind ($succ_0(v)$) des Wurzelknotens v von f_1.
   - der Kofaktor $f_1|_{x_i=1}$ dem 1-Kind ($succ_1(v)$) des Wurzelknotens v von f_1.

   Das gleiche gilt entsprechend für f_2.

Ein Algorithmus zur Berechnung von zweistelligen Booleschen Operationen (z.B. UND-Verknüpfung, ODER-Verknüpfung und Äquivalenz) mit ROBDDs ist der *APPLY*-Algorithmus [174]. Ein weiterer Algorithmus, der drei ROBDDs als Operanden bekommt und das Vorgehen des APPLY-Algorithmus auf drei ROBDDs als Operanden verallgemeinert, ist der *ITE*-Algorithmus [31, 106]. Er kann ebenfalls zur Berechnung von zweistelligen Booleschen Operationen benutzt werden. Die Art der Übergabe der drei ROBDDs an den ITE-Algorithmus bestimmt dabei, welche zweistellige Boolesche Operation berechnet wird.

Da in dieser Arbeit verbesserte Verfahren zur Bildberechnung bei der Modellprüfung vorgestellt werden und dort vor allem die zweistellige UND-Verknüpfung von ROBDDs verwendet wird, wird im Folgenden der zweistellige APPLY-Algorithmus genauer beschrieben. Pseudocode des APPLY-Algorithmus ist in Algorithmus 2 zu finden. Wie bereits

Algorithmus 2: APPLY-Algorithmus zur Berechnung von zweistelligen Booleschen Operationen $\circ$ mit ROBDDs.

```
1  APPLY(BoolOperation ∘, ROBDD F, ROBDD G)
2  if IsTerminalCase(∘, F, G) then
3  |  return TerminalCase(∘, F, G);
4  else
5  |  if ComputedTableHasEntry(∘, F, G) then
6  |  |  return ComputedTable(∘, F, G);
7  |  else
8  |  |  v = TopVariable(VariableRoot(F), VariableRoot(G));
9  |  |  R1 = APPLY(∘, F|v=1, G|v=1);
10 |  |  R0 = APPLY(∘, F|v=0, G|v=0);
11 |  |  if R0 == R1 then
12 |  |  |  return R1;
13 |  |  Result = FindOrAddUniqueTable(v, R0, R1);
14 |  |  InsertComputedTable(∘, F, G, Result);
15 |  |  return Result;
```

erwähnt, basiert er auf dem Durchlaufen der übergebenen ROBDDs durch rekursives Be-

rechnen von Kofaktoren bezüglich der jeweils obersten Variablen in der Variablenordnung, für die ein Knoten in einem der beiden ROBDDs vorhandenen ist. Zu Beginn des Algorithmus wird überprüft, ob im aktuellen Rekursionsschritt eine Terminierung der Berechnung für die übergebene Boolesche Operation $\circ$ möglich ist (Zeilen 2 und 3). Falls dies der Fall ist, kann die aktuelle rekursive Berechnung gestoppt werden und das Ergebnis der Operation $\circ$ für den aktuellen Rekursionsschritt wird berechnet und zurückgegeben. Bei der UND-Verknüpfung von ROBDDs ist eine Terminierung zum Beispiel möglich, wenn in einem Rekursionsschritt der Wurzelknoten eines der beiden ROBDDs ein Terminalknoten für den Booleschen Wert 0 ist. Das Ergebnis der UND-Verknüpfung für diesen Berechnungspfad ist dann der Wert 0 und der ROBDD, der nur aus dem Terminalknoten für den Booleschen Wert 0 besteht, wird als Ergebnis zurückgegeben. Wenn keine Terminierung möglich ist, wird anschließend geprüft, ob das Ergebnis der Booleschen Operation $\circ$ mit den aktuellen ROBDDs F und G bereits berechnet wurde und noch in der ComputedTable gespeichert ist (Zeile 5). Ist das Ergebnis der zu berechnenden Operation in der ComputedTable gespeichert, können die weiteren rekursiven Aufrufe für diesen Rekursionsschritt ebenfalls gestoppt werden und das in der ComputedTable vorhandene und bereits korrekt berechnete Ergebnis wird zurückgegeben (Zeile 6). Dies verhindert die mehrfache Neuberechnung von Operationen mit gleichen Operanden, was zu deutlichen Laufzeitverbesserungen führen kann (siehe auch Abschnitt 2.2.3). Da in der ComputedTable Ergebnisse mehrerer auch verschiedener Boolescher Operationen gespeichert werden können, werden neben dem Ergebnis einer Operation die Wurzelknoten der beteiligten ROBDDs F und G, sowie die Art der durchgeführten Booleschen Operation gespeichert. Damit wird sichergestellt, dass für eine Operation und deren Operanden nicht fälschlicherweise das Ergebnis einer anderen Berechnung zurückgegeben wird.

Konnte das Ergebnis für die durchzuführende Berechnung in der ComputedTable nicht gefunden werden, sind weitere rekursive Aufrufe des APPLY-Algorithmus notwendig. Dazu wird zuerst die oberste in der Variablenordnung vorkommende Variable v der Variablen der Wurzelknoten der ROBDDs F und G bestimmt (Zeile 8), die Top-Variable eines Rekursionsschritt genannt wird. Für diese Variable werden dann für beide ROBDDs die Kofaktoren bezüglich den Booleschen Werten 1 ($F|_{v=1}$, $G|_{v=1}$) und 0 ($F|_{v=0}$, $G|_{v=0}$) zur Benutzung in nachfolgenden rekursiven Aufrufen des Algorithmus berechnet, was effizient möglich ist (siehe Erklärungen weiter oben in diesem Abschnitt). Mit den resultierenden Kofaktoren wird der APPLY-Algorithmus erneut aufgerufen (Zeilen 9 und 10). Die Rückgabewerte der rekursiven Aufrufe sind die Ergebnisse der Berechnung der Booleschen Operation für die übergebenen Kofaktoren. Sind die beiden Ergebnisse gleich, müssen keine neuen Knoten erzeugt werden, da dann für beide Belegungen der Top-Variable des Rekursionsschritts das gleiche Ergebnis resultiert. Es kann das Ergebnis eines der beiden Aufrufe zurückgegeben werden (im Algorithmus R_1, siehe Zeile 12). Wenn sich die Ergebnisse der rekursiven Aufrufe unterscheiden, wird das Ergebnis des aktuellen Rekursionsschritts aus den Rückgabewerten R_0 und R_1 der rekursiven Aufrufe durch die Funktion *FindOrAddUniqueTable()* berechnet (Zeile 13). Pseudocode dieser Funktion ist in Algorithmus 3 zu finden.

Algorithmus 3: Funktion, die aus den Ergebnissen R_0 und R_1 der rekursiven Aufrufe des APPLY-Algorithmus den Rückgabewert des APPLY-Algorithmus für einen Rekursionsschritt berechnet.

```
1 FindOrAddUniqueTable(Variable v, ROBDD R_0, ROBDD R_1)
2 if IsInUniqueTable(v, R_0, R_1) then
3 |   return UniqueTable(v, R_0, R_1);
4 else
5 |   R = NewROBDDVertex(v, R_0, R_1);
6 |   InsertInUniqueTable(v, R_0, R_1, R);
7 |   return R;
```

Die Funktion prüft zuerst durch nachschauen in der UniqueTable, ob der zu erzeugende Ergebnisknoten bereits existiert (Zeile 2). Falls er existiert, wird der ROBDD mit diesem Knoten als Wurzelknoten zurückgegeben (Zeile 3) und es wird kein neuer Knoten erzeugt. Dadurch wird das mehrfache Erzeugen gleicher Knoten verhindert. Wenn der zu erzeugende Knoten noch nicht vorhanden ist, wird ein neuer Knoten für die Variable v mit den Kofaktoren R_0 und R_1 als Söhnen erzeugt (Zeile 5). Dadurch führt die Belegung der Booleschen Variable v für beide Booleschen Werte zum passenden Ergebnis der rekursiven Berechnung. Die Existenz des neuen Knotens wird danach in der UniqueTable vermerkt (Zeile 6). Der von der Funktion *FindOrAddUniqueTable()* in einem Rekursionsschritt zurückgegebene ROBDD ist das Resultat der Berechnung des APPLY-Algorithmus für diesen Rekursionsschritt. Das Ergebnis der Berechnung wird zur Beschleunigung weiterer späterer Berechnungen vom APPLY-Algorithmus noch in der ComputedTable abgespeichert (Zeile 14 in Algorithmus 2). Die Laufzeit des APPLY-Algorithmus liegt in $\mathcal{O}(|F| \cdot |G|)$, falls Einfügen und Anfrage für UniqueTable und ComputedTable in konstanter Zeit erfolgen können, wobei $|F|$ bzw. $|G|$ für die Anzahl der Knoten des jeweiligen ROBDDs steht.

2.3 Asynchrone nebenläufige Systeme

In der vorliegenden Arbeit werden Techniken zur Verbesserung der BDD-basierten Modellprüfung von endlichen asynchronen nebenläufigen Systemen präsentiert. Deshalb werden endliche asynchrone nebenläufige Systeme im Folgenden definiert.

Definition 2.26. (**Endliches asynchrones nebenläufiges System**) *Ein endliches asynchrones nebenläufiges System mit $m > 1$ Komponenten wird durch eine Kripke-Struktur $M^m = (S, S_0, R)$ beschrieben, mit:*

- *S, einer endlichen Menge von Zuständen. Zustände $s \in S$ sind Tupel $s = (\vec{g}, l_1, ..., l_m)$. Ein Zustand besteht aus den Werten von gemeinsamen globalen Variablen $\vec{g}$ und dem lokalen Zustand l_i jeder Komponente $i \in \{1, ..., m\}$. Der lokale Zustand einer Komponente setzt sich aus den Werten aller lokalen Variablen der Komponente zusammen.*

- $S_0 \subseteq S$, *der Menge der Anfangszustände.*
- $R \subseteq S \times S$ *der Transitionsrelation, die total sein muss. Das heißt,* $\forall s \in S\ \exists s' \in S$ *mit* $(s, s') \in R$. *Die Transitionsrelation setzt sich aus den Transitionsrelationen* $R_i = \{((\vec{g}, l_1, ..., l_m), (\vec{g}', l'_1, ..., l'_m)) | \textit{Zustandsübergang ist durch Ausführen einer Berechnung durch Komponente } i \textit{ möglich}\}$ *der einzelnen Komponenten* $i \in \{1, ..., m\}$ *zusammen:* $R = \bigcup_{i \in \{1,...,m\}} R_i$.
- *Das Ausführungsmodell eines solchen Systems ist das der verschachtelten Asynchronität, bei dem zu einem Zeitpunkt nur eine Komponente* $i \in \{1, ..., m\}$ *Transitionen aus ihrer Transitionsrelation* R_i *ausführen darf.*

In den asynchronen nebenläufigen Systemen nach Definition 2.26 ist es möglich, dass Transitionen einer Komponente von den lokalen Variablen anderer Komponenten abhängig sind. Ein Beispiel dafür ist das Abfragen der Werte von lokalen Variablen anderer Komponenten in Transitionen einer Komponente. Eine Einschränkung der Teile eines Systemzustands, von denen Transitionen einer Komponente abhängig sein dürfen, ist die Transitionslokalität. Diese wird in Definition 2.27 eingeführt. In endlichen asynchronen nebenläufigen Systemen mit Transitionslokalität hängen Transitionen einer Komponente nicht von den lokalen Variablen anderer Komponenten ab.

Definition 2.27. (**Transitionslokalität**) *Ein endliches asynchrones nebenläufiges System* $M^m = (S, S_0, R)$ *mit* $m > 1$ *Komponenten wird als transitionslokal bezeichnet, wenn Transitionen jeder Komponente* $i \in \{1, ..., m\}$ *nur von den Werten der globalen Variablen* $\vec{g}$ *und dem eigenen lokalen Zustand* l_i *der Komponente abhängen. Eine Komponente hat keinen Lese- und keinen Schreibzugriff auf lokale Variablen anderer Komponenten.*

Formal kann dies wie folgt beschrieben werden: Sei R_i *die Transitionsrelation einer Komponente* i *und* R_{i_P} *eine Relation, mit* $R_{i_P} = \{((\vec{g}, l_i), (\vec{g}', l'_i)) |$ $(\vec{g}', l'_i)$ *entsteht aus* $(\vec{g}, l_i)$ *durch Ausführen einer einzelnen Berechnung durch Komponente* $i\}$. *In Systemen mit Transitionslokalität gilt dann* $\forall i \in \{1, ..., m\} : R_i = \{((\vec{g}, l_1, ..., l_m), (\vec{g}', l'_1, ..., l'_m)) | \forall j \neq i :\ l'_j = l_j \wedge ((\vec{g}, l_i), (\vec{g}', l'_i)) \in R_{i_P}\}$ *und* $R = \bigcup_{i \in \{1,...,m\}} R_i$.

Ein Beispiel für asynchrone nebenläufige Systeme mit Transitionslokalität sind Programme für Mehrkern-Prozessoren, die aus parallel ausgeführten Abarbeitungsfäden (engl. Threads) bestehen, die nur über gemeinsame globale Variablen kommunizieren. Die Kommunikation über gemeinsame globale Variablen ist das bekannteste und flexibelste Modell der Kommunikation zwischen parallel ausgeführten Abarbeitungsfäden [124]. Es wird von bekannten Programmierschnittstellen (engl. APIs) zur Programmierung paralleler Programme (z.B. POSIX Threads oder Win64-API), die aufgrund der wachsenden Verbreitung von Mehrkern-Prozessoren immer wichtiger werden, unterstützt. Eine Teilmenge dieser Programme, die in der Praxis ebenfalls immer wichtiger wird, sind solche Programme aus replizierten Abarbeitungsfäden. Eine formale Definition dieser Programme ist in [124] angegeben.

Die symbolische Modellprüfung wird mit Mengen von Zuständen und Mengen von Transitionen durchgeführt. Eine Repräsentation von Kripke-Strukturen durch Boolesche charakteristische Funktionen bei binärer Kodierung von Zuständen und Transitionen wurde in

Abschnitt 2.1.5 vorgestellt. In der vorliegenden Arbeit werden Verbesserungen zur BDD-basierten symbolischen Modellprüfung bei binärer Kodierung präsentiert. Im Folgenden wird dazu auf die dafür verwendete Repräsentation asynchroner nebenläufiger Systeme, wie sie in Definition 2.26 eingeführt wurden, durch Boolesche charakteristische Funktionen eingegangen. Ein Zustand s eines asynchronen nebenläufigen Systems aus $m > 1$ Komponenten ist ein Tupel $s = (\vec{g}, l_1, ..., l_m)$, das aus dem globalen Zustand des Systems und den lokalen Zuständen der Komponenten besteht. Für eine binäre Kodierung von Systemzuständen ist hier eine binäre Kodierung der globalen Zustände und eine binäre Kodierung der lokalen Zustände jeder Komponente notwendig. Wenn die globalen Zustände mit n_g Bits und die lokalen Zustände einer Komponente i jeweils durch n_{l_i} Bits kodiert werden können, so kann ein Systemzustand durch $N = n_g + \sum_{i=1}^{m} n_{l_i}$ Bits kodiert werden. Zur binären Kodierung einer Transition müssen jeweils der Start- und der Zielzustand der Transition binär kodiert werden. Daher sind dafür $2 \cdot N$ Bits notwendig. Damit können Zustandsmengen für solche Systeme durch charakteristische Funktionen mit N Booleschen Eingabevariablen und Mengen von Transitionen durch charakteristische Funktionen mit $2 \cdot N$ Booleschen Eingabevariablen repräsentiert werden. Ein einfaches Beispiel eines asynchronen nebenläufigen Systems mit Transitionslokalität, das im weiteren Verlauf der vorliegenden Arbeit noch häufiger für Beispiele verwendet wird, ist in Abschnitt 2.6 angegeben. In diesem Abschnitt sind zusätzlich BDDs zur partitionierten Repräsentation der Transitionsrelation (siehe auch Abschnitt 2.5.2) und zur Repräsentation des Anfangszustands des Systems angegeben.

2.4 Ausnutzung von Symmetrien bei der Modellprüfung

Ein zentrales Problem bei der Modellprüfung ist das Problem der Zustandsraumexplosion. Die Zahl der Zustände eines Systems steigt exponentiell mit der Anzahl der Variablen und auch exponentiell mit der Anzahl der in einem System vorhandenen Komponenten [8]. Daher tritt das Problem der Zustandsraumexplosion besonders stark bei der Verifikation nebenläufiger Systeme mit vielen Komponenten auf.

Nebenläufige Systeme besitzen aber oft eine Menge von Symmetrien. Dies gilt besonders für solche Systeme mit replizierten Komponenten, die in der Praxis häufig vorkommen. Beispiele sind Sensornetzwerke mit hunderten von identisch implementierten Sensorknoten oder Client-Server Architekturen mit identisch implementierten Clients. Zur Ausnutzung von Symmetrien bei der Modellprüfung wurden Symmetriereduktionstechniken entwickelt (siehe z.B. [55, 56, 77, 113, 116, 147, 193]). Durch deren Anwendung können häufig signifikante Reduktionen des Speicherbedarfs und der Laufzeit erreicht werden. Im Folgenden werden Grundlagen zur Ausnutzung von Symmetrien durch replizierte Komponenten in nebenläufigen Systemen bei der Modellprüfung behandelt. Dies ist die bei der Modellprüfung am besten untersuchte und am weitesten verbreitete Symmetrieart. Neben Symmetrien durch replizierte Komponenten gibt es aber noch weitere Arten von Symmetrien. Eine davon ist die Datensymmetrie [115, 122, 123, 196]. Sie tritt auf, wenn verschiedene Werte von Variablen bei der Verifikation von Eigenschaften ununterscheidbar sind. In solchen Fällen kann die Modellprüfung unter Verwendung der für die Verifikationsaufgabe unter-

scheidbaren Werte dieser Variablen durchgeführt werden. Ein Beispiel für das Auftreten von Datensymmetrien ist zum Beispiel die Verifikation von Cache-Kohärenz Protokollen für Mehrprozessorsysteme. Die in den beteiligten Caches gespeicherten Datenwerte sind dort häufig nicht wichtig, um die korrekte Funktionsweise der Protokolle zu überprüfen. Ebenso wurden Techniken zur Ausnutzung von Symmetrien für Systeme entwickelt, die nur teilweise oder temporär Symmetrien aufweisen (siehe z.B. [82, 78, 188, 192]).

Symmetriereduktionstechniken versuchen das redundante Durchmustern von äquivalenten Teilen des Zustandsraums bei der Modellprüfung zu vermeiden. Sie nutzen vorhandene Symmetrien aus, in dem sie die Durchmusterung des Zustandsraums auf Repräsentanten von Äquivalenzklassen von Zuständen beschränken. Dadurch wird versucht, den zu durchmusternden Zustandsraum zu verkleinern und damit das Problem der Zustandsraumexplosion abzumildern. Die Hauptprobleme bei der Anwendung von Symmetriereduktionstechniken sind die Entscheidung, ob Zustände unter vorhandenen Symmetrien zur gleichen Äquivalenzklasse gehören und die Erkennung von Symmetrien. Neben den in den folgenden Abschnitten zur Symmetriereduktion bei der Modellprüfung aufgeführten Informationen finden sich Zusammenfassungen zur Symmetriereduktion z.B. in [56, 147, 193].

2.4.1 Grundlagen zur Symmetriereduktion

Eine Kripke-Struktur ist symmetrisch, wenn sie invariant gegenüber bestimmten Transformationen ihres Zustandsraums ist. In den folgenden Abschnitten werden Symmetrien für Kripke-Strukturen definiert und eine durch deren Ausnutzung resultierende symmetriereduzierte Quotienten Kripke-Struktur eingeführt. Für diese wurde gezeigt, dass sie die gleichen symmetrischen CTL*-Eigenschaften erfüllt, wie eine entsprechende unreduzierte Kripke-Struktur. Die Modellprüfung kann bei vorhandenen Symmetrien mit der symmetriereduzierten Quotienten Kripke-Struktur, die aus Repräsentanten von Äquivalenzklassen von unter Symmetrien äquivalenten Zuständen besteht, durchgeführt werden.

2.4.1.1 Permutationen und Gruppen

Symmetrien von Kripke-Strukturen können durch Permutationen beschrieben werden [185]. Für eine gegebene nichtleere Menge X ist eine Permutation auf X eine bijektive Abbildung $\pi : X \to X$, die jedem Element der Menge X ein Element der gleichen Menge zuordnet. Mit der Hintereinanderausführung als Operation bildet die Menge aller Permutationen auf einer Menge X eine Gruppe.

Definition 2.28. (Gruppe, [56]) *Eine Gruppe ist eine Menge G zusammen mit einer binären Operation $\circ : G \times G \to G$, für die gilt:*

- *Für alle α, β, $\gamma \in G$ gilt: $\alpha \circ (\beta \circ \gamma) = (\alpha \circ \beta) \circ \gamma$.*
- *Es gibt ein neutrales Element $e \in G$, so dass für alle $\alpha \in G$ gilt: $\alpha = e \circ \alpha = \alpha \circ e$.*
- *Zu jedem Gruppenelement $\alpha \in G$ gibt es ein inverses Element $\alpha^{-1} \in G$, mit $\alpha \circ \alpha^{-1} = \alpha^{-1} \circ \alpha = e$.*

Weitere Grundlagen zur Gruppentheorie finden sich z.B. in [42, 108, 170].

In einem nebenläufigen System aus n replizierten Komponenten besteht ein Systemzustand $s = (\vec{g}, l_1, ..., l_n)$ aus einem globalen Zustand $\vec{g}$, der durch die Werte der vorhandenen gemeinsamen globalen Variablen bestimmt wird, und dem lokalen Zustand l_i jeder Komponente. Dies entspricht dem Systemtyp, der in Abschnitt 2.3 eingeführt wurde. Es existiert lediglich die Einschränkung, dass Systeme in der vorliegenden Arbeit im Zusammenhang mit Symmetriereduktion aus replizierten Komponenten bestehen. Für ein System aus n replizierten Komponenten können Symmetrien durch replizierte Komponenten aus Permutationen über der Menge $\{1, 2, ..., n\}$ abgeleitet werden. Diese Permutationen können zur Definition von Permutationen $\pi : S \to S$ über der Zustandsmenge $s \in S$ einer Kripke-Struktur benutzt werden. Dazu wird die Wirkung einer Permutation π über $\{1, 2, ..., n\}$ auf einen Zustand $s = (\vec{g}, l_1, ..., l_n)$ der Kripke-Struktur durch $\pi(s) = (\vec{g}_\pi, l_{\pi(1)}, ..., l_{\pi(n)})$ definiert. Damit spezifiziert eine Permutation π durch Angabe von Vertauschungen von Komponentenindizes das Vertauschen von lokalen Zuständen von Komponenten. Der globale Zustand resultiert aus den Werten der vorhandenen globalen Variablen. Die globalen Variablen sind in $\vec{g}$ zusammengefasst dargestellt. Für m globale Variablen mit $m \geq 1$ ist $\vec{g} = (g_1, ..., g_m)$ und es gilt $\vec{g}_\pi = (g_1^\pi, ..., g_m^\pi)$. Es gibt zwei Arten von globalen Variablen, id-sensitive globale Variablen und id-insensitive globale Variablen. Die Wirkung von π auf eine globale Variable g_i hängt von der Art der globalen Variablen ab. Id-sensitive Variablen zeigen auf Komponentenindizes. Wenn eine Permutation die Vertauschung eines Komponentenindexes angibt, auf den eine globale id-sensitive Variable zeigt, muss der Wert der id-sensitiven Variablen bei Anwendung der Permutation auf einen Zustand gemäß der Permutation vertauscht werden. Dies ist notwendig, da auch eine entsprechende Vertauschung bei den lokalen Zuständen der Komponenten stattfindet.

In Abschnitt 2.1.2 wurde ein Protokoll eingeführt, das in einem nebenläufigen System mit mehreren Komponenten den wechselseitigen Ausschluss für einen kritischen Abschnitt sicherstellt. Abbildung 2.3 zeigt ein Zustandsübergangsdiagramm des Protokolls für eine Komponente. Für drei Komponenten K_i ($i \in \{1, 2, 3\}$), ist zum Beispiel der Zustand $s = (2, N, K, V)$ ein in diesem Protokoll möglicher Zustand. In diesem Zustand ist K_1 im lokalen Zustand N ($l_1 = N$), K_2 im lokalen Zustand K ($l_2 = K$) und K_3 im lokalen Zustand V ($l_3 = V$). Die vorhandene globale Variable ist eine id-sensitive Variable, die auf Komponentenindizes zeigt. Sie zeigt im Zustand s auf den Index 2 für die Komponente K_2. Sei nun für dieses Beispiel eine Permutation von Komponentenindizes π durch

$$\pi = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 1 & 3 \end{pmatrix}$$

gegeben. In der Matrix steht in der ersten Zeile die zu permutierende natürliche Zahl j und in der zweiten Zeile der zugehörige Funktionswert $\pi(j)$ der Permutation. Für diese Permutation gilt damit: $\pi(1) = 2$, $\pi(2) = 1$ und $\pi(3) = 3$. Durch Anwendung der Permutation auf den Zustand s ergibt sich $\pi(s) = \pi((2, N, K, V)) = (1, K, N, V)$. Sie bewirkt, gemäß der angegebenen Vertauschungen von Komponentenindizes, die Vertauschung der lokalen Zustände der Komponenten K_1 und K_2. Außerdem wird der Wert der globalen id-sensitiven Variablen entsprechend angepasst. Diese zeigt nach Anwendung der Permu-

tation weiterhin auf die Komponente, die sich gerade im kritischen Abschnitt (in ihrem lokalen Zustand K) befindet.

Id-insensitive globale Variablen sind keine Zeiger auf Komponentenindizes. Ihre Werte ändern sich durch Anwendung einer Permutation nicht, d.h. es gilt für jede id-insensitive Variable g_i: $g_i^\pi = g_i$. Ein Beispiel für eine solche Variable ist ein binäres Semaphor, das in einem nebenläufigen System mit mehreren Komponenten zur Synchronisation des Zugriffs auf einen kritischen Abschnitt verwendet wird. Wenn es den Wert *TRUE* besitzt, darf eine beliebige Komponente unabhängig vom Komponentenindex den kritischen Abschnitt betreten.

Sind in einem System keine id-sensitiven Variablen vorhanden, werden durch Anwendung von Permutationen nur lokale Zustände von Komponenten vertauscht. Dies ist für eine Permutation auch der Fall, wenn ein System bei vorhandenen globalen id-sensitiven Variablen in einem Zustand ist, für den die Permutation keine Vertauschung der Werte der globalen id-sensitiven Variablen angibt. Ein Beispiel dafür für das Protokoll aus Abschnitt 2.1.2 und drei Komponenten ist die Anwendung der Permutation

$$\pi = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \end{pmatrix}$$

auf den Zustand $s = (2, N, K, V)$. Durch Anwendung der Permutation auf den Zustand s ergibt sich $\pi(s) = \pi((2, N, K, V)) = ((2, V, K, N))$. Die Permutation π bildet den Index der Komponente 2 auf sich selbst ab ($\pi(2) = 2$). Daher wird der Wert der globalen id-sensitiven Variablen durch Anwendung der Permutation nicht verändert. Es werden nur die lokalen Zustände der Komponenten K_1 und K_3 gemäß der Permutation vertauscht. Neben globalen id-sensitiven Variablen gibt es auch lokale id-sensitive Variablen. Dies sind id-sensitive Variablen unter den lokalen Variablen der Komponenten, deren Werte den lokalen Zustand von Komponenten bestimmen. Auf diese können Permutationen ebenso wie auf globale id-sensitive Variablen Auswirkungen haben. Für die in dieser Arbeit durchgeführten Verifikationsexperimente wurden keine Testfälle mit lokalen id-sensitiven Variablen verwendet. Daher werden diese hier nicht näher betrachtet. Weitere Informationen über lokale id-sensitive Variablen bei der Symmetriereduktion finden sich z.B. in [193].

2.4.1.2 Symmetrien und Kripke-Strukturen

In Abschnitt 2.4.1.1 wurde beschrieben, wie Permutationen $\pi : S \to S$ über Zuständen einer Kripke-Struktur für ein System aus n replizierten Komponenten aus Permutationen über der Menge $\{1, 2, ..., n\}$ abgeleitet werden können. Die Wirkung von Permutationen auf Transitionen $\pi : R \to R$, kann bei Benutzung der Definition der Wirkung von Permutationen π auf Zustände für eine Transition $(s, t) \in R$ durch $\pi((s, t)) = (\pi(s), \pi(t))$ definiert werden.

Definition 2.29. (Symmetrien einer Kripke-Struktur, [185]) *Eine Permutation π über S wird Symmetrie einer Kripke-Struktur $M = (S, S_0, R, L)$ genannt, wenn:*

- *R invariant unter π ist: $\pi(R) = R$.*
- *S_0 invariant unter π ist: $\pi(S_0) = S_0$.*
- *L invariant unter π ist: $L(s) = L(\pi(s))$ für jeden Zustand $s \in S$.*

Die Symmetrien einer Kripke-Struktur M bilden mit der Hintereinanderausführung als Operation eine Gruppe G. Eine Kripke-Struktur M wird symmetrisch genannt, wenn ihre Symmetriegruppe G nicht trivial ist (d.h. die Symmetriegruppe nicht nur aus der Permutation besteht, die jedes Element auf sich selbst abbildet (Identitätspermutation)).

Eine Gruppe G von Symmetrien einer Kripke-Struktur M erzeugt auf der Zustandsmenge S der Kripke-Struktur eine Äquivalenzrelation $\equiv_G$. Dabei gilt für zwei Zustände $s, t \in S : s \equiv_G t \Leftrightarrow s = \pi(t)$ für ein $\pi \in G$. Die Äquivalenzklasse $[s]_G$ eines Zustands $s \in S$ unter der Äquivalenzrelation $\equiv_G$ ist die Menge $[s]_G = \{t \mid t \in S \wedge \exists \pi \in G : \pi(s) = t\}$, die Orbit von s bezüglich G genannt wird. Die Relation $\equiv_G$ wird auch Orbit-Relation genannt. Aus Definition 2.29 folgt, dass wenn für zwei Zustände $s, t \in S$ einer Kripke-Struktur $s \equiv_G t$ für eine Symmetriegruppe G gilt, dann gilt auch $L(s) = L(t)$. Dies gilt, da L invariant unter der Anwendung von Permutationen der Symmetriegruppe G ist. Die Orbits der Zustandsmenge S einer Kripke-Struktur bezüglich einer Symmetriegruppe G können zur Definition einer Quotienten Kripke-Struktur benutzt werden.

Definition 2.30. (Quotienten Kripke-Struktur, [147]) *Die Quotienten Kripke-Struktur M_G von M bezüglich einer Symmetriegruppe G ist ein Quadrupel $M_G = (S_G, S_G^0, R_G, L_G)$ mit:*

- *$S_G = \{[s]_G : s \in S\}$, die Menge der Orbits von S bezüglich G.*
- *$S_G^0 = \{[s]_G : s \in S_0\}$, die Orbits der Anfangszustände S_0 bezüglich G.*
- *$R_G = \{([s]_G, [t]_G) : (s, t) \in R\}$, die Quotienten Transitionsrelation.*
- *$L_G([s]_G) = L(rep_G([s]_G))$, wobei $rep_G([s]_G)$ ein eindeutiger Repräsentant von $[s]_G$ ist.*

Bei der praktischen Anwendung der Symmetriereduktion wird üblicherweise statt der Menge S_G von Orbits ein Zustand jedes Orbits als Repräsentant des Orbits ausgewählt. Damit wird die Quotienten Kripke-Struktur M_G mit Zuständen gebaut, die in der unreduzierten Kripke-Struktur M ebenfalls vorhanden sind. Dies hat praktische Vorteile, wie zum Beispiel die Benutzung der gleichen Zustandskodierung für die Quotienten Kripke-Struktur M_G und die unreduzierte Kripke-Struktur M.

Falls die Symmetriegruppe G nicht nur aus der Permutation für die Identität besteht, besitzt die Quotienten Kripke-Struktur M_G weniger Zustände als M. Für einen Zustand $s \in S$ einer Kripke-Struktur ist die Größe seines Orbits $[s]_G$ durch die Größe der Symmetriegruppe $|G|$ beschränkt. Dadurch ergibt sich eine theoretische Minimalgröße der

Zustandsmenge S_G von M_G von $|S|/|G|$. In Systemen mit vielen Symmetrien kann für die Mächtigkeit von G gelten $|G| = n!$, wobei n die Anzahl der Komponenten im System ist.

Es gibt verschiedene Arten von Symmetrien. In dieser Arbeit wird die Ausnutzung von Symmetrien durch replizierte Komponenten betrachtet. Eine in solchen Systemen häufig vorkommende Art von Symmetrien ist die vollständige Symmetrie. Bei ihr sind in Systemzuständen beliebige Permutationen der Zustände der replizierten Komponenten erlaubt. Dadurch ist bei vollständiger Symmetrie, im Vergleich zu anderen Symmetriegruppen, oft eine größere Reduktion des Speicherbedarfs erreichbar. Eine weitere häufig vorkommende Art von Symmetrien ist die Rotationssymmetrie. Sie existiert, wenn die replizierten Komponenten in einem Ring angeordnet sind und Rotationen des Rings von Komponenten (bzw. der entsprechenden lokalen Komponentenzustände) zu Systemzuständen, die die gleichen symmetrischen Eigenschaften erfüllen, führen.

Für vollständige Symmetrie ist ein Beispiel einer Quotienten Kripke-Struktur für das Beispielprotokoll aus Abschnitt 2.1.2, das den wechselseitigen Ausschluss für einen kritischen Abschnitt in einem nebenläufigen System mit mehreren Komponenten sicherstellt, für zwei Komponenten in Abbildung 2.8 dargestellt. Eine entsprechende unreduzierte Kripke-Struktur ist in Abbildung 2.2 zu finden. Die Repräsentanten der Orbits wurden für die Quotienten Kripke-Struktur so ausgewählt, dass die lokalen Zustände der Komponenten in alphabetischer Ordnung gemäß $K < N < V$ aufsteigend sortiert sind. Vorhandene globale id-sensitive Variablen wurden bei gleichem lokalen Zustand der Komponenten auf den kleinsten möglichen Wert gesetzt. Allgemeine Informationen zur Auswahl von Repräsentanten finden sich in Abschnitt 2.4.1.3. Anstatt 12 Zuständen besitzt die symmetriereduzierte Quotienten Kripke-Struktur nur noch 6 Zustände.

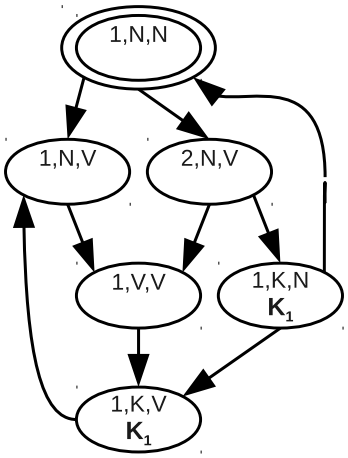

Abbildung 2.8: Quotienten Kripke-Struktur zur unreduzierten Kripke-Struktur aus Abbildung 2.2, für das Protokoll zur Sicherstellung des wechselseitigen Ausschlusses aus Abschnitt 2.1.2.

Es kann gezeigt werden, dass die Kripke-Strukturen M und M_G die gleiche Menge an CTL*-Eigenschaften, die invariant unter den Vertauschungen von G sind, erfüllen. Ein Beweis des folgenden Satzes 2.31 ist in [55] angegeben.

Satz 2.31. ([55]) *Sei $M = (S, S_0, R, L)$ eine Kripke-Struktur, G eine Symmetriegruppe von M und h eine CTL*-Formel. Wenn die in h vorkommenden atomaren Eigenschaften invariant unter der Anwendung von Symmetrien der Gruppe G sind, gilt*

$$M, s \models h \Leftrightarrow M_G, [s]_G \models h \tag{2.7}$$

wobei M_G die Quotienten Kripke-Struktur von M bezüglich G ist.

Damit kann die Modellprüfung für eine CTL*-Formel bei Auswahl einer geeigneten Symmetriegruppe G, unter der die CTL*-Formel invariant ist, mit der Quotienten Kripke-Struktur M_G anstatt mit M durchgeführt werden. Dies gilt auch für die Erreichbarkeitsanalyse und die Überprüfung von symmetrischen Invarianten, da diese durch CTL*-Formeln dargestellt werden können (siehe Abschnitt 2.1.6.3). Die Verwendung der Quotienten Kripke-Struktur kann zu einer beträchtlichen Reduktion des Speicherbedarfs und auch zu Laufzeitverringerungen führen (siehe z.B. [55]).

2.4.1.3 Auswahl von Orbit-Repräsentanten

Bei vorhandenen Symmetrien und der Verwendung von Symmetriereduktionstechniken, kann die Modellprüfung für symmetrische Eigenschaften mit der symmetriereduzierten Quotienten Kripke-Struktur durchgeführt werden. Wie im letzten Abschnitt beschrieben, werden Äquivalenzklassen $[s]_G$ von Zuständen einer Kripke-Struktur bezüglich einer Gruppe G von Symmetrien Orbits genannt. In der praktischen Anwendung der Symmetriereduktion bringt es Vorteile die Quotienten Kripke-Struktur aus Repräsentanten von Orbits zu bauen. Eine der wichtigsten Operationen dabei und bei der Benutzung von Symmetriereduktionstechniken ist die Berechnung dieser Orbit-Repräsentanten. Während der Exploration der Quotienten Kripke-Struktur kann für einen gerade explorierten Zustand mit Hilfe der Berechnung des entsprechenden Orbit-Repräsentanten entschieden werden, ob vorher bereits ein Zustand mit für die Verifikationsaufgabe äquivalenten Eigenschaften exploriert wurde. Dies kann durch Vergleich des berechneten Orbit-Repräsentanten mit den bereits vorher explorierten und abgespeicherten Repräsentanten entschieden werden. Wurde der berechnete Repräsentant bereits vorher gefunden, braucht der Repräsentant nicht erneut zur Weiterexploration gespeichert werden. Der Grund ist, dass der entsprechende Orbit-Repräsentant bereits vorher zur Weiterexploration vorgemerkt wurde.

Für beliebige Symmetriegruppen wurde gezeigt, dass das Problem der Entscheidung, ob zwei Zustände zum gleichen Orbit gehören, so schwer wie das Problem der Berechnung von Graph-Isomorphismen ist [57]. Dafür ist kein Algorithmus in Polynomialzeit bekannt. Das Problem der Entscheidung ob zwei Zustände zum gleichen Orbit gehören wird auch das *Orbit-Problem* genannt. Für viele häufig vorkommende Symmetriegruppen kann das Orbit-Problem aber effizient gelöst werden. Beispiele dafür sind die in Abschnitt 2.4.1.2 eingeführte vollständige Symmetrie und die Rotationssymmetrie [57].

Ein praktischer und gewöhnlich zur Auswahl von Repräsentanten von Orbits verwendeter Ansatz ist die Verwendung der lexikografisch kleinsten Zustände der Orbits als Repräsentanten. Eine lexikografische Ordnung $<_{Lex}$ auf den Zuständen $s = (\vec{g}, l_1, ..., l_n) \in S$ einer Kripke-Struktur kann aus einer totalen Ordnung auf der Menge der lokalen Zustände von Komponenten und einer lexikografischen Ordnung über der Menge der globalen Zustände $<_{Lex_g}$ abgeleitet werden [193].

Definition 2.32. (Lexikografische Ordnung für Zustände einer Kripke-Struktur) *Gegeben seien zwei Zustände $s, s' \in S$ einer Kripke-Struktur mit $s = (\vec{g}, l_1, ..., l_n)$ und $s' = (\vec{g}', l'_1, ..., l'_n)$. Zustand s ist lexikografisch kleiner als s', $s <_{Lex} s'$, wenn*

- $(l_1, ..., l_n) <_{Lex} (l'_1, ..., l'_n)$, *oder*
- $\forall i \in \{1, ..., n\}$ *gilt* $l_i = l'_i$ *und* $\vec{g} <_{Lex_g} \vec{g}'$.

Wenn die lokalen Zustände der Komponenten für zwei Zustände gleich sind, werden in Definition 2.32 zusätzlich die Werte der globalen Variablen berücksichtigt. Dabei ist $\vec{g}$, wie in Abschnitt 2.4.1.1 beschrieben, ein Vektor aus id-insensitiven und id-sensitiven globalen Variablen. Die Werte von id-sensitiven Variablen sind Zeiger auf Komponentenindizes. Auf deren Werte kann die Anwendung von Permutationen Auswirkungen haben. Id-insensitive Variablen verändern sich durch Anwendung von Permutationen nicht und besitzen für alle Zustände eines Orbits die gleichen Werte. Um bei gleichem lokalen Zustandsvektor dennoch einen eindeutigen lexikografisch kleinsten Repräsentanten zu erhalten, wird eine lexikografische Ordnung ($<_{Lex_g}$) für den globalen Zustandsvektor $\vec{g}$ verwendet. Die Werte von id-sensitiven globalen Variablen werden für den Repräsentanten eines Orbits durch die lexikografische Ordnung $<_{Lex_g}$ so ausgewählt, dass sie jeweils minimale Werte besitzen.

Da in der vorliegenden Arbeit keine Verifikationsexperimente mit lokalen id-sensitiven Variablen durchgeführt wurden, werden hier nur globale id-sensitive Variablen behandelt und Definition 2.32 gilt nur für Systeme ohne lokale id-sensitive Variablen. Informationen zu lokalen id-sensitiven Variablen und weitere Informationen über Orbit-Repräsentanten bei vorhandenen id-sensitiven Variablen, sind in [68, 193] zu finden. Für das Beispiel des Protokolls zur Sicherstellung des wechselseitigen Ausschlusses für zwei Komponenten aus Abschnitt 2.1.2 ist eine Quotienten Kripke-Struktur aus Orbit-Repräsentanten in Abbildung 2.8 zu sehen. Als totale Ordnung für die lokalen Zustände der Komponenten wurde hier die Ordnung $K < N < V$ verwendet.

Es gibt effiziente Verfahren zur Berechnung von Repräsentanten für einzelne Zustände bei vorhandener vollständiger Symmetrie oder bei Rotationssymmetrie. Wenn keine id-sensitiven Variablen vorhanden sind, kann bei vollständiger Symmetrie der lexikografisch kleinste Zustand eines Orbits durch Sortierung der lokalen Zustände der Komponenten im Zustandsvektor berechnet werden [57]. Dies ist mit einer Komplexität von $\mathcal{O}(n \log n)$ möglich, wobei n die Anzahl der im System vorhandenen Komponenten ist. Sind globale id-sensitive Variablen vorhanden, müssen deren Werte noch entsprechend angepasst werden. Ein Verfahren, das bei kleinen Symmetriegruppen mit wenigen Permutationen zur Repräsentantenberechnung verwendet werden kann, ist die Bestimmung des lexikografisch

kleinsten Zustands eines Orbits durch Anwenden aller Permutationen. Für große Symmetriegruppen wäre dieses Vorgehen hingegen zu aufwendig. Ein Beispiel für eine Symmetriegruppe mit wenigen Permutationen ist die Rotationssymmetrie. In einem System mit n Komponenten gibt es für einen Zustand n verschiedene Rotationen der lokalen Komponentenzustände. Der lexikografisch kleinste Zustand kann für die Rotationssymmetrie daher in linearer Zeit durch Anwenden aller Permutationen bestimmt werden [57].

Ausführlicher und auch für andere als die oben genannten Symmetriegruppen, wird die Berechnung von Orbit-Repräsentanten zum Beispiel in [29, 68, 69] behandelt. Die oben genannten Verfahren zur Repräsentantenberechnung können bei der expliziten Modellprüfung, bei der mit einzelnen Zuständen gearbeitet wird, direkt angewendet werden. Bei der symbolischen Modellprüfung mit BDDs muss zusätzlich beachtet werden, dass mit Mengen von Zuständen gearbeitet wird, für die die Repräsentanten auf einmal berechnet werden müssen.

2.4.1.4 Detektion von Symmetrien

Damit Symmetriereduktionstechniken bei der Modellprüfung angewendet werden können, müssen in den zu prüfenden Systemen vorhandene Symmetrien entdeckt werden. Ein möglicher einfacher Ansatz zur Ermittlung vorhandener Symmetrien ist es zuerst die unreduzierte Kripke-Struktur M zu bauen, um danach die in ihr vorhandenen Symmetrien durch Benutzung von Standard-Algorithmen zum Berechnen von Graph-Isomorphismen zu bestimmen. Dies kann mit Hilfe von Programmen zum Bestimmen von Graph-Isomorphismen, wie zum Beispiel *nauty* [141] oder *Traces* [141], durchgeführt werden. Die erkannte Symmetriegruppe kann dann benutzt werden, um die Quotienten Kripke-Struktur M_G zu bauen, wodurch angegebene Eigenschaften über dieser Kripke-Struktur geprüft werden können. Allerdings ist die Berechnung von Graph-Isomorphismen für große Kripke-Strukturen aufwendig und es muss bei diesem Verfahren die unreduzierte Kripke-Struktur M gebaut werden. Damit kann einer der größten Vorteile von Symmetriereduktionstechniken nicht ausgenutzt werden. Dieser ist die Durchführung der Modellprüfung mit einer symmetriereduzierten Kripke-Struktur, auch wenn die unreduzierte Kripke-Struktur zu viel Speicherbedarf erfordert. Um die Möglichkeiten der Symmetriereduktion bestmöglich nutzen zu können, wurden daher Techniken zur Detektion von Symmetrien entwickelt, die den Bau der unreduzierten Kripke-Struktur nicht benötigen.

Häufig versuchen Techniken zur Erkennung vorhandener Symmetrien diese im Eingabeprogramm zu erkennen. Ein Ansatz dazu ist die Eingabesprache des Modellprüfers zu beschränken, so dass nur Systeme mit einer bestimmten Art von Symmetrien spezifiziert werden können. Meistens muss dort die Existenz mehrerer Komponenten eines Komponententyps und die für diese Komponenten vorhandene Symmetrie in der Eingabe des Modellprüfers angegeben werden. Die vorhandene Symmetriegruppe kann dann leicht bestimmt werden. Dieser Ansatz wurde zum Beispiel bei den Modellprüfern SMC [178] und Sviss [191, 190] gewählt. Beim Modellprüfer Murphi [116] wurde die Eingabesprache um einen speziellen Datentyp *Skalarset* erweitert. Der Wertebereich dieses Datentyps ist ein Integer-Bereich und es ist nur eine beschränkte Menge von Operationen mit diesem Datentyp möglich. Mit Hilfe der Restriktionen wird sichergestellt, dass konsistente

Permutationen der Werte von Skalarset-Variablen in einem Zustand zu unter Symmetrien äquivalenten Zuständen führen. Dieser Ansatz wird ebenfalls vom Symmetriepaket SymmSPIN [30], das den Modellprüfer SPIN [1] um Symmetriereduktionstechniken erweitert, benutzt.

Andere Ansätze versuchen vorhandene Symmetrien aus der aus einem Eingabeprogramm ermittelten Kommunikationsstruktur zwischen Komponenten eines Systems zu berechnen. Dies erfolgt durch die Berechnung von Graph-Isomorphismen für die Kommunikationsstruktur. Der Graph für die Kommunikationsstruktur ist häufig klein und die zugrundeliegende Symmetriegruppe kann wieder mit Programmen zur Berechnung von Graph-Isomorphismen herausgefunden werden. Unter bestimmten Bedingungen können daraus Symmetrien abgeleitet werden, die auch in der Kripke-Struktur vorhanden sind und die während der Modellprüfung ausgenutzt werden können [57, 77, 121]. Dieses Vorgehen wird im Paket TopSPIN [65] verfolgt, das ebenfalls implementiert wurde um den Modellprüfer SPIN um Symmetriereduktionstechniken zu erweitern. Der Vorteil dieser Verfahren ist, dass sie häufig für eine größere Menge an Symmetriearten als die oben genannten Verfahren, bei denen Symmetrien in der Eingabesprache angegeben werden, anwendbar sind. Weitere Informationen zur Erkennung von vorhandenen Symmetrien finden sich zum Beispiel in [64, 193].

2.4.2 Verfahren zur Symmetriereduktion

Das Ziel von Symmetriereduktionstechniken bei der Modellprüfung ist es, den zu durchmusternden Zustandsraum durch Ausnutzen von in Systemen vorhandenen Symmetrien zu verkleinern. Damit kann das Problem der Zustandsraumexplosion abgemildert werden. In Abschnitt 2.4.1.2 wurde beschrieben, dass die Modellprüfung von CTL*-Eigenschaften, die invariant unter einer Symmetriegruppe G einer Kripke-Struktur M sind, mit einer Quotienten Kripke-Struktur aus Repräsentanten von Orbits durchgeführt werden kann. Ein üblicherweise verwendeter Ansatz zur Auswahl der dazu nötigen Repräsentanten von Orbits wurde in Abschnitt 2.4.1.3 beschrieben. In den folgenden Abschnitten werden konkrete Symmetriereduktionstechniken vorgestellt, durch deren Anwendung die Modellprüfung mit einer Quotienten Kripke-Struktur durchgeführt wird.

2.4.2.1 Symmetriereduktion bei der expliziten Modellprüfung

In diesem Abschnitt wird die Anwendung von Symmetriereduktionstechniken bei der expliziten Modellprüfung (siehe auch Abschnitt 2.1.4) beschrieben. Dort werden einzelne Zustände und Transitionen explizit abgespeichert. Der Einbau von Symmetriereduktionstechniken erfolgt meistens durch Modifikationen bei der Berechnung von Vorgänger- oder Nachfolgerzuständen. Da für die Verifikationsexperimente in der vorliegenden Arbeit die symbolische Vorwärtserreichbarkeitsanalyse mit BDDs (siehe auch Abschnitt 2.1.6.3) verwendet wurde, wird in diesem Abschnitt die Verwendung von Symmetriereduktionstechniken bei der expliziten Modellprüfung auch am Beispiel der Vorwärtserreichbarkeitsanalyse beschrieben. Mit dieser können alle von den Anfangszuständen einer Kripke-Struktur erreichbaren Zustände berechnet werden.

Der im Folgenden angegebene Algorithmus 4 exploriert alle von den Anfangszuständen einer Kripke-Struktur erreichbaren symmetriereduzierten Zustände. In diesen Algorithmus kann der Test der Gültigkeit von Invarianten für neu explorierte Zustände direkt integriert werden [116]. Damit muss bei nicht erfüllten Invarianten der Zustandsraum nur so weit aufgebaut werden, bis festgestellt werden kann, dass eine Invariante nicht erfüllt ist (sogenannte *on-the-fly Modellprüfung*, siehe Abschnitt 2.1.4). Aus Gründen der besseren Verständlichkeit wird der Algorithmus hier aber ohne integrierte Überprüfung von Invarianten eingeführt.

Als Modifikation zu einem entsprechenden Algorithmus ohne Ausnutzung von Symmetrien wird eine sogenannte Kanonisierungsfunktion $rep_G : S \to S$ benutzt. Diese berechnet für einen Zustand $s \in S$ einer Kripke-Struktur den äquivalenten symmetriereduzierten Repräsentanten $rep_G(s)$ des Orbits des Zustands. Eindeutige Repräsentanten von Orbits werden benutzt, um die Äquivalenz von Zuständen unter den vorhandenen Symmetrien zu entscheiden. Der angegebene Algorithmus verwendet eine Tabelle mit bereits gefundenen

Algorithmus 4: Explizite Vorwärtserreichbarkeitsanalyse ausgehend von den Startzuständen S_0 einer Kripke-Struktur mit Ausnutzung von Symmetrien.

```
1 Reached = {rep_G(s) : s ∈ S_0}
2 Unexplored = {rep_G(s) : s ∈ S_0}
3 while Unexplored ≠ ∅ do
4 |   Entferne einen Zustand s aus Unexplored
5 |   forall the Nachfolger q von s do
6 |   |   if rep_G(q) ∉ Reached then
7 |   |   |   füge rep_G(q) zu Reached hinzu
8 |   |   |   füge rep_G(q) zu Unexplored hinzu
```

Zuständen *Reached* und eine Liste aus bisher noch nicht weiter explorierten Zuständen *Unexplored*. Um die Kripke-Struktur nur aus symmetriereduzierten Zuständen aufzubauen, werden die beiden Datenstrukturen zu Beginn mit den Repräsentanten der Orbits der Anfangszustände der Kripke-Struktur initialisiert (Zeilen 1 und 2). Danach wird die Schleife in Zeile 3 solange ausgeführt wie noch Zustände, deren Nachfolger noch nicht exploriert wurden, vorhanden sind. In einer Iteration der Schleife werden alle Nachfolgerzustände eines bisher noch unexplorierten Zustands exploriert. Um für einen Nachfolgerzustand q eines Zustands s zu überprüfen, ob ein unter Symmetrien äquivalenter Zustand bereits exploriert wurde, wird mit Hilfe der Kanonisierungsfunktion $rep_G(q)$ der Repräsentant des Orbits dieses Zustands berechnet (Zeile 6). Wurde noch kein symmetrieäquivalenter Zustand exploriert, wird der Repräsentant $rep_G(q)$ zu *Reached* und *Unexplored* hinzugefügt und dadurch später weiterexploriert. Wurde schon ein unter den vorhandenen Symmetrien äquivalenter Zustand exploriert, muss der Nachfolgerzustand q nicht weiter exploriert werden.

Der Unterschied von Algorithmus 4 zum entsprechenden Algorithmus ohne Symmetriereduktion ist die zur Berechnung von Orbit-Repräsentanten verwendete Kanonisierungs-

funktion rep_G. Die Komplexität des Algorithmus hängt direkt von der Komplexität der Berechnung der Repräsentanten ab, weshalb die effiziente Durchführung der Repräsentantenberechnung sehr wichtig ist. Leider ist die Berechnung von Repräsentanten für beliebige Symmetriegruppen aber so schwer wie die Berechnung von Graph-Isomorphismen. Da dieses Problem in der praktischen Anwendung manchmal mit verträglichem Aufwand lösbar ist, wurden schon explizite Modellprüfer entwickelt, die für die Berechnung von Repräsentanten Programme zur Berechnung von Graph-Isomorphismen aufrufen [182, 186]. Außerdem gibt es Symmetriegruppen, für die die Repräsentantenberechnung effizient möglich ist. Für die vollständige Symmetrie und die Rotationssymmetrie wurden entsprechende, für die explizite Modellprüfung direkt anwendbare Ansätze in Abschnitt 2.4.1.3 präsentiert. Um den Aufwand zur Berechnung von eindeutigen Orbit-Repräsentanten zu reduzieren wurden auch andere Strategien, wie zum Beispiel die Verwendung mehrerer Repräsentanten für einen Orbit [29, 30, 57, 66], entwickelt.

2.4.2.2 Symmetriereduktion bei der symbolischen Modellprüfung

Bei der symbolischen Modellprüfung mit BDDs werden Mengen von Zuständen und Mengen von Transitionen durch BDDs repräsentiert (siehe Abschnitte 2.1.4 und 2.1.5). Mit Bildberechnungen zur Ermittlung von Nachfolgerzuständen, werden hier bei der Vorwärtserreichbarkeitsanalyse alle aus einer gegebenen Menge von Zuständen durch eine Menge von Transitionen erreichbaren Nachfolgerzustände ermittelt. Für die symbolische Modellprüfung mit BDDs benötigt der naheliegendste und in [55] vorgestellte Ansatz zur Exploration einer symmetriereduzierten Quotienten Kripke-Struktur M_G, anstatt der entsprechenden unreduzierten Kripke-Struktur M, eine Repräsentation der Orbit-Relation (siehe Abschnitt 2.4.1.2) als BDD. Nach dem Aufbau des BDDs für die Orbit-Relation kann ein BDD für eine Kanonisierungsfunktion $rep_G : S \to S$ berechnet werden, der Zustände der Kripke-Struktur auf eindeutige Repräsentanten (z.B. lexikografisch kleinste Zustände) der entsprechenden Orbits abbildet. Mit Hilfe des BDDs der Kanonisierungsfunktion kann aus der Transitionsrelation der unreduzierten Kripke-Struktur die Quotienten-Transitionsrelation R_G über Repräsentanten von Orbits

$$R_G(x,y) = \{([s]_G,[t]_G) : (s,t) \in R \wedge rep_G(s) = [s]_G \wedge rep_G(t) = [t]_G\} \tag{2.8}$$

der Quotienten Kripke-Struktur berechnet werden. Die Quotienten-Transitionsrelation kann anschließend bei der Modellprüfung für die Berechnung von Nachfolgerzuständen benutzt werden. Dadurch werden nur symmetriereduzierte Nachfolgerzustände exploriert und es kann die Zustandsmenge S_G der Quotienten Kripke-Struktur berechnet werden.

In [55] wurde aber gezeigt, dass der BDD für die Orbit-Relation für viele häufig verwendete Symmetriegruppen (unter anderem für die vollständige Symmetrie) eine intraktable Größe besitzt. Für vollständige Symmetrie und ein System aus n Komponenten mit jeweils l lokalen Zuständen wurde eine untere Schranke für die BDD-Größe der Orbit-Relation bewiesen, die exponentiell im Minimum der Anzahl von Komponenten und der Anzahl der lokalen Zustände einer Komponente ist. Daher können Ansätze zur Symmetriereduktion, die die Orbit-Relation als BDD benötigen, nur für Systeme mit wenigen Komponenten

oder mit Komponenten mit wenigen lokalen Zuständen verwendet werden. Es gibt auch Verfahren bei denen die Quotienten-Transitionsrelation ohne die Orbit-Relation erzeugt wird. Allerdings ist der BDD der Quotienten-Transitionsrelation häufig selbst sehr groß und kann daher in der Regel nur für sehr kleine Systeme gebaut werden [76, 193]. Der Grund dafür ist, dass die Orbit-Relation im wesentlichen darin eingebettet ist.

Da der BDD der Orbit-Relation häufig eine intraktable Größe besitzt, wurden Methoden zur symbolischen Modellprüfung mit BDDs mit Ausnutzung von Symmetrien entwickelt, die den BDD der Orbit-Relation nicht explizit benötigen. Der *mehrere Repräsentanten*-Ansatz (engl. multiple representatives) verwendet statt einem Repräsentanten mehrere Repräsentanten für jeden Orbit [55, 57]. Dabei muss von jedem Orbit mindestens ein Repräsentant vorhanden sein und ein Zustand kann mehreren Repräsentanten zugeordnet sein. Die Idee bei diesem Verfahren ist, dass der BDD zur Repräsentation einer Teilrelation der Orbit-Relation kleiner sein könnte. Allerdings wird bei Verwendung mehrerer Repräsentanten für jeden Orbit nicht mehr die maximale durch Symmetriereduktion zu erzielende Speicherplatzreduktion erreicht. In [9, 10] wird ein *On-the-fly Repräsentantenauswahl* (engl. On-the-fly Representative Selection) genannter Ansatz vorgestellt, bei dem die Orbit-Relation ebenfalls nicht benötigt wird. Es findet dort auch keine Berechnung von Repräsentanten von Orbits statt. Als Orbit-Repräsentanten werden die Zustände eines Orbits ausgewählt, die bei der Exploration des Zustandsraums als erste als Ergebnis einer Bildberechnung auftreten. Werden beim ersten Auftreten eines Zustands eines Orbits mehrere Zustände des Orbits gleichzeitig exploriert, liefert dieses Verfahren mehrere Repräsentanten für diesen Orbit. Im schlimmsten Fall könnten auch alle Zustände eines Orbits als Repräsentanten ausgewählt werden. Dafür ist aber keine separate Berechnung von Orbit-Repräsentanten nötig. Die Bildberechnungen werden mit der unveränderten Transitionsrelation der Kripke-Struktur durchgeführt. Um zu entscheiden, ob bereits ein Zustand eines Orbits exploriert wurde, werden alle zu den gespeicherten Repräsentanten symmetrischen Zustände berechnet und aus den mit einer Bildberechnung ermittelten Nachfolgerzuständen entfernt. Die übrig bleibenden Nachfolgerzustände sind Zustände von Orbits, für die vorher noch kein Zustand exploriert wurde. Sie werden als neue Repräsentanten gespeichert. Ein Problem dieses Verfahrens ist, dass die Berechnung der BDDs, in denen alle zu den bisherigen Repräsentanten gespeicherten symmetrischen Zustände enthalten sind, aufwendig und die resultierenden BDDs sehr groß sein können. Außerdem wird bei diesem Verfahren bei mehreren Repräsentanten für einen Orbit, wie beim mehrere Repräsentanten-Ansatz, nicht die maximal mögliche Symmetriereduktion erreicht. Um dieses Problem abzumildern, wird in [9, 10] die Verwendung einer Unterapproximation der Menge der weiter zu explorierenden Zustände empfohlen.

Eine weitere Methode, die die Orbit-Relation nicht benötigt und bei der keine Repräsentanten von Orbits berechnet werden müssen, ist die Benutzung von *generischen Repräsentanten* (engl. generic representatives) [78]. Sie kann bei der expliziten und bei der symbolischen Modellprüfung angewendet werden. Die Symmetriereduktion findet hier durch eine Übersetzung des ursprünglichen Eingabeprogramms in ein symmetriereduziertes Programm und eine entsprechende Anpassung der zu prüfenden Eigenschaften statt. Danach kann die Verifikation ohne weitere Berücksichtigung von Symmetrien mit üblichen

Modellprüfungsalgorithmen durchgeführt werden. Für ein System ohne globale Variablen aus n Komponenten mit m lokalen Zuständen und vorhandener vollständiger Symmetrie, werden durch die Übersetzung des Eingabeprogramms Zustände $s = (l_1, ..., l_n)$ zu Zuständen $s = (n^l_1, ..., n^l_m)$ des reduzierten Programms. Ein Zustand des reduzierten Programms ist ein Vektor von Zählern, wobei bei m lokalen Zuständen einer Komponente für jeden lokalen Zustand ein Zähler n^l_i ($i \in \{1, ..., m\}$) vorhanden ist. Dieser Zähler gibt für einen lokalen Zustand LZ die Anzahl der Komponenten K_i an, die in einem Systemzustand in diesem lokalen Zustand sind: $n^l_{LZ} = \#\{K_i | i \in \{1, ..., n\} \wedge l_i = LZ\}$. In einem System ohne globale Variablen wird bei vollständiger Symmetrie jeder Zustand eines Orbits auf den gleichen Vektor von Zählern abgebildet. Der Vektor repräsentiert damit den gesamten Orbit. Es ergibt sich durch das reduzierte Programm eine zur Quotienten Kripke-Struktur isomorphe Kripke-Struktur, die Vektoren aus Zählern als Zustände besitzt. Der beschriebene Ansatz ist auch unter dem Namen *Zählerabstraktion* (engl. counter abstraction) [162] bekannt. Seine Anwendbarkeit wurde in [75] auf Systeme mit id-insensitiven und id-sensitiven globalen Variablen erweitert. Die dort präsentierten experimentellen Ergebnisse zeigen, dass die Benutzung von generischen Repräsentanten zu besseren Ergebnissen als die Verwendung von eindeutigen Repräsentanten unter Benutzung der Orbit Relation, oder das Verfahren mit mehreren Repräsentanten führt. Ein Nachteil der Zählerabstraktion ist das Problem der Explosion der Anzahl der lokalen Zustände (engl. local state explosion problem). Komponenten können oft eine sehr große Anzahl an lokalen Zuständen besitzen und für jeden lokalen Zustand muss ein Zähler vorgesehen werden. Bei Verwendung symbolischer Datenstrukturen, wie zum Beispiel BDDs, macht dies durch die nötige Vorreservierung von Bits für die Kodierung von Zählern Probleme. Verfahren zur Abmilderung dieses Problems werden in [13, 14, 79] präsentiert. Außerdem ist die Anwendbarkeit des Ansatzes auf wenige Symmetriegruppen, aber unter anderem für die häufig vorkommende vollständige Symmetrie, beschränkt.

2.4.2.3 Dynamische symbolische Symmetriereduktion

Die dynamische Symmetriereduktion [76] ist eine Symmetriereduktionstechnik für die symbolische Modellprüfung mit BDDs. Bei ihr werden eindeutige Orbit-Repräsentanten von symbolischen Modellprüfungsalgorithmen während Fixpunktberechnungen dynamisch berechnet. Durch die dynamische Berechnung von Repräsentanten wird der Bau des häufig intraktabel großen BDDs mit der Orbit-Relation nicht benötigt. Aufgrund der Verwendung von eindeutigen Orbit-Repräsentanten wird trotzdem die maximal mögliche Symmetriereduktion erzielt. Außerdem werden im Gegensatz zu den in Abschnitt 2.4.2.2 präsentierten Ansätzen mit der Orbit-Relation und dem mehrere Repräsentanten-Ansatz nur Repräsentanten für Orbits gespeichert, für die auch Zustände bei Berechnungen auftreten. Im Vergleich mit der Benutzung von generischen Repräsentanten ist keine möglicherweise komplizierte Übersetzung des Eingabeprogramms in ein symmetriereduziertes Programm notwendig. Die dynamische Symmetriereduktion kann für viele bekannte Symmetriegruppen angewendet werden. Neben der Symmetrie durch replizierte Komponenten kann sie auch zur Ausnutzung von Datensymmetrien (siehe Abschnitt 2.4) benutzt werden. In [76] werden Ergebnisse von Verifikationsexperimenten präsentiert, bei denen die dynamische

Symmetriereduktion mit dem mehrere Repräsentanten-Ansatz und der Verwendung von generischen Repräsentanten verglichen wird. Die dynamische Symmetriereduktion führt dort bei allen durchgeführten Verifikationsexperimenten zu besseren Ergebnissen als die anderen beiden Verfahren.

Da für die Verifikationsexperimente in dieser Arbeit die symbolische Vorwärtserreichbarkeitsanalyse verwendet wurde, wird die dynamische Symmetriereduktion in diesem Abschnitt für die symbolische Vorwärtserreichbarkeitsanalyse mit BDDs eingeführt. Eine Verallgemeinerung des Prinzips der dynamischen Symmetriereduktion für die CTL-Modellprüfung ist in [76] zu finden. In Abschnitt 2.1.6.3 wurde ein Algorithmus zur symbolischen Vorwärtserreichbarkeitsanalyse eingeführt. Eine Erweiterung dieses Algorithmus um Symmetriereduktion mit Benutzung der Quotienten-Transitionsrelation R_G ist in Algorithmus 5 dargestellt. Diese wird dort in Zeile 4 zur Berechnung von Nachfolgerzuständen benutzt, wodurch nur symmetriereduzierte Zustände exploriert werden. Wie in Abschnitt 2.4.2.2 beschrieben, kann die Quotienten-Transitionsrelation in der Regel aber nur für sehr kleine Systeme gebaut werden [76, 193].

Algorithmus 5: Symbolische Vorwärtserreichbarkeitsanalyse ausgehend von den Startzuständen S_0 einer Kripke-Struktur mit Symmetriereduktion durch Benutzung der Quotienten-Transitionsrelation R_G.

```
1 Reached = S_0
2 ToExplore = S_0
3 while ToExplore ≠ ∅ do
4 |   ExploredStates = Successors_{R_G}(ToExplore)
5 |   Unexplored = ExploredStates \ Reached
6 |   ToExplore = Unexplored
7 |   Reached = Reached ∪ Unexplored
```

Die dynamische Symmetriereduktion benötigt den Bau der Orbit-Relation und auch den Bau der Quotienten-Transitionsrelation nicht. In Algorithmus 6 ist der Algorithmus zur symbolischen Vorwärtserreichbarkeitsanalyse mit Verwendung der dynamischen Symmetriereduktion, anstatt der Benutzung der Quotienten-Transitionsrelation, zu sehen. Der einzige Unterschied zu Algorithmus 5 mit der Quotienten-Transitionsrelation ist bei der Berechnung von Nachfolgerzuständen. Diese werden bei Verwendung der dynamischen Symmetriereduktion mit der unreduzierten Transitionsrelation R einer Kripke-Struktur durchgeführt. Danach werden die explorierten Nachfolgerzustände dynamisch durch Anwendung einer Abstraktionsfunktion α auf Repräsentanten von Orbits abgebildet ($\alpha(\text{Successors}_R(\text{ToExplore}))$, siehe Zeile 4).

Eine formale Definition der Abstraktionsfunktion α ist in Gleichung 2.9 angegeben. Darin ist $\equiv_G$ die in Abschnitt 2.4.1.2 eingeführte Orbit-Relation. In Abhängigkeit der vorhandenen Symmetrieart muss die Implementierung von α angepasst werden. Durch entsprechende Auswahl der Implementierung von α kann Algorithmus 6 zur Anwendung

Algorithmus 6: Symbolische Vorwärtserreichbarkeitsanalyse ausgehend von den Startzuständen S_0 einer Kripke-Struktur mit Symmetriereduktion durch Benutzung der dynamischen Symmetriereduktion.

```
1 Reached = S_0
2 ToExplore = S_0
3 while ToExplore ≠ ∅ do
4 |   ExploredStates = α(Successors_R(ToExplore))
5 |   Unexplored = ExploredStates \ Reached
6 |   ToExplore = Unexplored
7 |   Reached = Reached ∪ Unexplored
```

der dynamischen Symmetriereduktion bei verschiedenen Symmetriearten benutzt werden.

$$\alpha(T) = \{rep_G([t]_G) \in S_G : \exists t \in T : (t, rep_G([t]_G)) \ \in \equiv_G\} \tag{2.9}$$

Ein für viele Symmetriegruppen praktischer Ansatz ist die Wahl des lexikografisch kleinsten Zustands eines Orbits als dessen Orbit-Repräsentant. Der lexikografisch kleinste Zustand wird dabei gemäß einer lexikografischen Ordnung $<_{Lex}$ auf den Zuständen $s = (\vec{g}, l_1, ..., l_n) \in S$ einer Kripke-Struktur bestimmt (siehe auch Abschnitt 2.4.1.3). In [76, 193] und im Folgenden wird die Implementierung der Abstraktionsfunktion α für die Berechnung von lexikografisch kleinsten Zuständen von Orbits als Repräsentanten vorgestellt. Da die Verifikationsexperimente mit dynamischer Symmetriereduktion in dieser Arbeit fast alle mit vollständiger Symmetrie durchgeführt wurden (eine Ausnahme sind die Experimente mit dem Philosophenproblem, bei dem Rotationssymmetrie vorhanden ist), wird die Abstraktionsfunktion α hier für die in der Praxis am häufigsten vorkommende vollständige Symmetrie vorgestellt. Eine Verallgemeinerung auf andere Symmetriegruppen, z.B. die Rotationssymmetrie, ist in [76] zu finden.

Besitzt ein System keine globalen id-sensitiven Variablen, kann der lexikografisch kleinste Zustand eines Orbits unter vollständiger Symmetrie durch Sortierung der lokalen Zustände der Komponenten ermittelt werden. Die symbolische Modellprüfung mit BDDs arbeitet mit Mengen von Zuständen und mit Mengen von Transitionen anstatt mit einzelnen Zuständen und Transitionen, wie die explizite Modellprüfung. Um die lexikografisch kleinsten Zustände durch Sortierung bestimmen zu können, müssen daher nicht einzelne Zustände, sondern Mengen von Zuständen gleichzeitig sortiert werden. Allerdings besitzen Systeme häufig globale id-sensitive Variablen, deren Werte Komponentenindizes entsprechen. Diese müssen bei der Berechnung des lexikografisch kleinsten Zustands eines Orbits ebenfalls berücksichtigt werden. Eine Lösung dafür ist es Systemzustände mit sortierten lokalen Zuständen der Komponenten als Repräsentanten zu verwenden, bei denen die globalen id-sensitiven Variablen minimale Werte besitzen (siehe Abschnitt 2.4.1.3).

Bei der dynamischen Symmetriereduktion werden Mengen von Zuständen durch Vertauschen von Variablen zur Kodierung der lokalen Zustände von Komponenten in BDDs simultan sortiert. Sind globale id-sensitive Variablen vorhanden, werden zusätzlich noch

deren Werte angepasst. Der Aufwand zur Vertauschung von Variablen, der für ein Variablenpaar durch gegenseitiges ineinander Umbenennen der Variablen des Variablenpaares durchgeführt werden kann, dominiert die Effizienz der dynamischen Symmetriereduktion. Dabei hängt der Aufwand der Vertauschungsoperation exponentiell von der Entfernung der zu vertauschenden Variablen in der Variablenordnung eines BDDs ab. Seien zum Beispiel x_i und x_j zwei Variablen, die Distanz d zueinander in der Variablenordnung haben und es kommt x_i vor x_j in der Variablenordnung vor. Dann gibt es für jedes Auftreten eines Knotens für die Variable x_i, aufgrund der für Variablen die in einer Variablenordnung zwischen x_i und x_j liegen möglichen verschiedenen Variablenbelegungen, höchstens 2^d Nachfolgerknoten die x_j repräsentieren. Bei Vertauschung der Variablen durch gegenseitiges Umbenennen, muss dies für alle 2^d Nachfolgerpfade auf denen mit x_j markierte Knoten vorkommen können durchgeführt werden. Führt für eine Eingabebelegung und ein solches Variablenpaar die Variablenbelegung $x_i = 0$ und $x_j = 1$ vor der Umbenennung der Variablen zu einem nachfolgenden Teil-BDD, wird dieser Teil-BDD nach der Vertauschung der Variablen bei gleichbleibendem restlichen Teil der Eingabebelegung durch die Variablenbelegung $x_i = 1$ und $x_j = 0$ erreicht. Dies entspricht einem Vertauschen der durch die Eingabebelegungen erreichten Teil-BDDs, was in Abbildung 2.9 an einem aus [189] entnommenen Beispiel veranschaulicht wird.

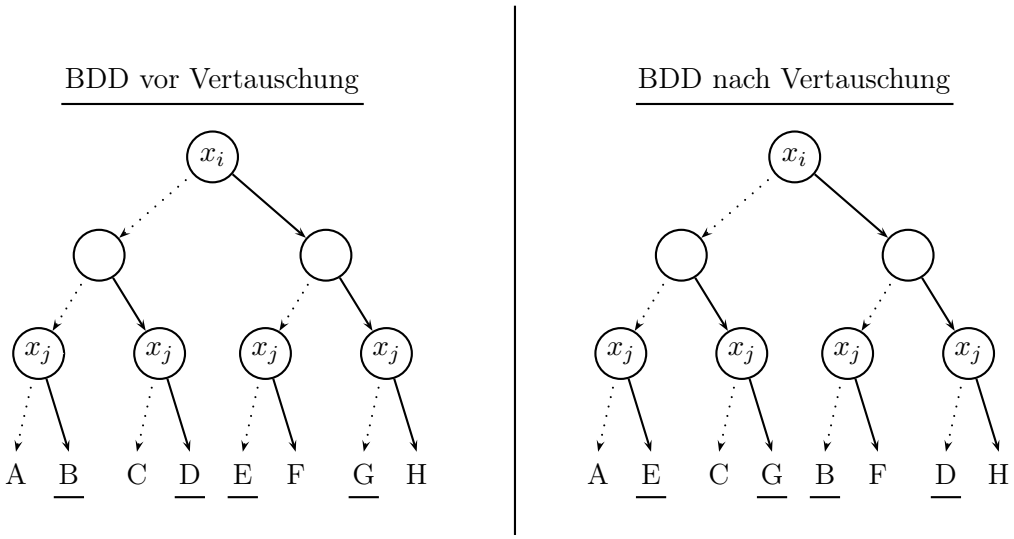

Abbildung 2.9: Veranschaulichung der Vertauschung von zwei Variablen x_i und x_j durch Umbenennen. Dadurch werden Teilbaum B mit Teilbaum E und Teilbaum D mit Teilbaum G vertauscht.

Da die Lokalität der zu vertauschenden Variablen einen großen Einfluss auf die Effizienz der dynamischen Symmetriereduktion hat, wird in [76] empfohlen Bubble Sort zur Sortierung der Zustandsmengen zu verwenden. Bubble Sort hat den Vorteil, dass nur Variablen für lokale Zustände benachbarter Komponenten vertauscht werden. Wird die Variablenordnung der BDDs so gewählt, dass die Entfernung zwischen Variablen für lokale Zustände

benachbarter Komponenten möglichst klein ist, kann mit Bubble Sort der Aufwand für die Vertauschungsoperationen klein gehalten werden.

Algorithmus 7 zeigt für vollständige Symmetrie eine Implementierung der Abstraktionsfunktion $\alpha(T)$ zur symbolischen Sortierung einer übergebenen Zustandsmenge. In Zeile 4 des Algorithmus wird die Funktion $\tau(Z)$ aufgerufen. Pseudocode dieser Funktion ist in Algorithmus 8 zu finden. Sie führt, falls nötig, eine Umsortierung von lokalen Zuständen benachbarter Komponenten durch. Die Funktion $\tau(Z)$ wird so lange aufgerufen, bis ein Fixpunkt erreicht ist und der BDD Z mit den Orbit-Repräsentanten zur im BDD T gespeicherten Zustandsmenge berechnet ist.

Algorithmus 7: Abstraktionsfunktion $\alpha(T)$ zur Berechnung von eindeutigen Repräsentanten mit dynamischer Symmetriereduktion bei vollständiger Symmetrie. Diese ruft solange Algorithmus 8 auf, bis alle Repräsentanten für Zustände der übergebenen Zustandsmenge T berechnet wurden.

```
1 Z = T;
2 repeat
3 |  Z' = Z;
4 |  Z = τ(Z);
5 until (Z == Z');
6 return Z;
```

Die Implementierung der Funktion $\tau(Z)$ wird in Algorithmus 8 beschrieben. Sie vergleicht lokale Zustände benachbarter Komponenten und berücksichtigt auch die Werte von globalen id-sensitiven Variablen. Falls nötig werden die Booleschen Variablen, die benutzt werden um die lokalen Zustände benachbarter Komponenten zu repräsentieren, vertauscht und die Werte von globalen id-sensitiven Variablen angepasst. $\leq_z$ ist in Algorithmus 8 für einen festen Zustand z einer Kripke-Struktur eine totale Ordnung über den Komponenten. Für Zustände einer Kripke-Struktur wurde in Abschnitt 2.4.1.3 eine lexikografische Ordnung $<_{Lex}$ eingeführt. Aus dieser kann die totale Ordnung $\leq_z$ abgeleitet werden, die hier aus Gründen der besseren Verständlichkeit ohne Berücksichtigung von id-sensitiven Variablen angegeben wird. Dafür bedeutet $p \leq_z p+1$ für einen Zustand z, zwei Komponenten p, $p+1$ und deren lokale Zustände $l_p(z)$ und $l_{p+1}(z)$ im Zustand z, dass $l_p(z) \leq l_{p+1}(z)$ gilt. Damit ist die Menge der Repräsentanten für n Komponenten mit $\leq_z$ definiert durch:

$$rep_G(S) = \{z \in S : \forall p < n : p \leq_z p+1\} = \bigcap_{p<n} \{z \in S : p \leq_z p+1\}. \tag{2.10}$$

Algorithmus 8: Funktion $\tau(Z)$, die die Zustände benachbarter Komponenten vergleicht und diese vertauscht, falls sie die Ordnung $\leq_z$ verletzen.

```
1 for p = 1; p ≤ (n −1); p++ do
2     Z_bad = Z ∧ ¬{z : p ≤_z p + 1};
3     if Z_bad ≠ ∅ then
4         Z_good = Z \ Z_bad;
5         Z_swapped = swap(p, p+1, Z_bad);
6         Z = Z_good ∨ Z_swapped;
7 return Z;
```

Bei jedem Aufruf von Algorithmus 8 wird für alle benachbarten Komponenten geprüft, ob die Ordnung $\leq_z$ verletzt ist, was innerhalb der für die Komponentenindizes bis zum Komponentenindex $n-1$ ausgeführten *for*-Schleife durchgeführt wird (Zeile 1). In einer Iteration der *for*-Schleife wird dafür für die aktuell betrachteten benachbarten Komponenten p und $p+1$ die Menge der Zustände in der übergebenen Zustandsmenge Z ermittelt, die die Ordnung $\leq_z$ verletzen (Zeile 2). Diese Zustände werden im BDD Z_{bad} gespeichert. Wenn die mit dem BDD Z_{bad} repräsentierte Zustandsmenge nicht leer ist, werden die nötigen Vertauschungen der Booleschen Variablen für die benachbarten Komponenten für alle mit diesem BDD repräsentierten Zustände, wie oben beschrieben durch Aufruf der Funktion *swap(p, p+1, Z_{bad})*, simultan durchgeführt (Zeile 5). Das Resultat einer Iteration der *for*-Schleife setzt sich aus den Zuständen, in denen die betrachteten lokalen Zustände der Komponenten bereits die richtige Ordnung hatten (Z_{good}) und aus den Zuständen die aus durchgeführten Vertauschungen resultieren ($Z_{swapped}$) zusammen. Da in Algorithmus 8 nur Vergleiche für benachbarte Komponenten durchgeführt werden, wird dieser von Algorithmus 7 zur Repräsentantenberechnung solange aufgerufen, bis für eine übergebene Zustandsmenge keine Anpassungen mehr nötig sind und die Repräsentanten berechnet wurden.

2.4.2.4 Zustandssymmetrien

In den vorangehenden Abschnitten wurden Symmetrien von Kripke-Strukturen behandelt. Neben diesen gibt es bei der Modellprüfung von nebenläufigen Systemen mit replizierten Komponenten auch Symmetrien innerhalb einzelner Zustände $s \in S$ einer Kripke-Struktur. Diese werden als Zustandssymmetrien bezeichnet [77, 105, 178]. Bisher wurden Zustandssymmetrien nur für die explizite Modellprüfung untersucht. In der vorliegenden Arbeit wird in Abschnitt 7.3 ein Ansatz zur Ausnutzung von Zustandssymmetrien bei der symbolischen dynamischen Symmetriereduktion mit BDDs präsentiert. Die in Abschnitt 8.6 aufgeführten Ergebnisse von damit durchgeführten Verifikationsexperimenten zeigen, dass durch die Verwendung von Zustandssymmetrien auch bei der symbolischen Modellprüfung große Laufzeitverbesserungen erzielt werden können.

Zustandssymmetrien resultieren aus äquivalenten Eigenschaften von Komponentenzuständen in einem Zustand einer Kripke-Struktur durch zwischen Komponenten vorhandene Symmetrien.

Definition 2.33. (Äquivalenz in einem Zustand) *Zwei Komponenten i und j eines asynchronen nebenläufigen Systems sind äquivalent in einem Zustand $s \in S$ einer Kripke-Struktur des Systems, wenn es eine Permutation π mit $\pi(i) = j$ gibt, deren abgeleitete Permutation über der Zustandsmenge $\pi : S \to S$ eine Symmetrie ist und wenn für diese $\pi(s) = s$ gilt.*

Die Definition von Permutationen $\pi : S \to S$ aus Permutationen über der Menge $\{1, 2, ..., n\}$ für ein System aus n replizierten Komponenten wurde in Abschnitt 2.4.1.1 beschrieben. Symmetrien einer Kripke-Struktur wurden in Abschnitt 2.4.1.2 eingeführt. Aus Äquivalenzen in einem Zustand lässt sich eine Partition der Komponenten ableiten.

Definition 2.34. (Zustandssymmetriepartition) *Die Relation Äquivalenz in einem Zustand $s \in S$ einer Kripke-Struktur ist eine Äquivalenzrelation über der Menge der Komponenten eines asynchronen nebenläufigen Systems. Die Äquivalenzklassen bilden eine Partition auf der Menge der Komponenten. Diese Partition wird Zustandssymmetriepartition von s genannt.*

Alle Komponenten einer Zustandssymmetriepartition eines Zustands können in diesem Zustand die gleichen Transitionen ausführen. Bei Anwendung von Symmetriereduktionsmethoden ergeben sich daraus bei der Berechnung von Repräsentanten für Nachfolgerzustände eines solchen Zustands für alle Komponenten einer Zustandssymmetriepartition die gleichen eindeutigen Repräsentanten. Zustandssymmetrien können dort ausgenutzt werden, in dem jeweils nur für eine repräsentative Komponente einer Zustandssymmetriepartition Transitionen ausgeführt werden. Dadurch werden alle Repräsentanten von Nachfolgerzuständen exploriert, aber redundante zu gleichen Repräsentanten führende Repräsentantenberechnungen werden vermieden. Dies kann zu großen Laufzeitverbesserungen führen.

In Abbildung 2.10 ist ein Beispiel für die Nachfolgerberechnung bei vorhandenen Zustandssymmetrien für das in Abschnitt 2.1.2 vorgestellte Protokoll, das den wechselseitigen Ausschluss in einem System mit mehreren Komponenten sicherstellt, dargestellt. Im Beispiel sind fünf Komponenten vorhanden und es besteht vollständige Symmetrie zwischen den Komponenten. Der obere Teil der Abbildung veranschaulicht die Exploration ohne Ausnutzung von Zustandssymmetrien. Dort ist ein Ausgangszustand $s = (1, N, N, N, N, N)$, bestehend aus dem Wert der vorhandenen globalen id-sensitiven Variablen (1) und den lokalen Zuständen der Komponenten (jeweils der lokale Zustand N) angegeben. Mit diesem Ausgangszustand wird dann die Nachfolgerberechnung für alle Komponenten ausgeführt, wodurch fünf Nachfolgerzustände exploriert werden. Für die Nachfolgerzustände werden anschließend die Repräsentanten nach der in Abschnitt 2.4.1.3 eingeführten lexikografischen Ordnung für Zustände berechnet (für die lokalen Zustände gilt hier $N < V$). Im unteren Teil der Abbildung ist die Exploration unter Ausnutzung von Zustandssymmetrien zu sehen. Die Zustandssymmetriepartition des Zustands

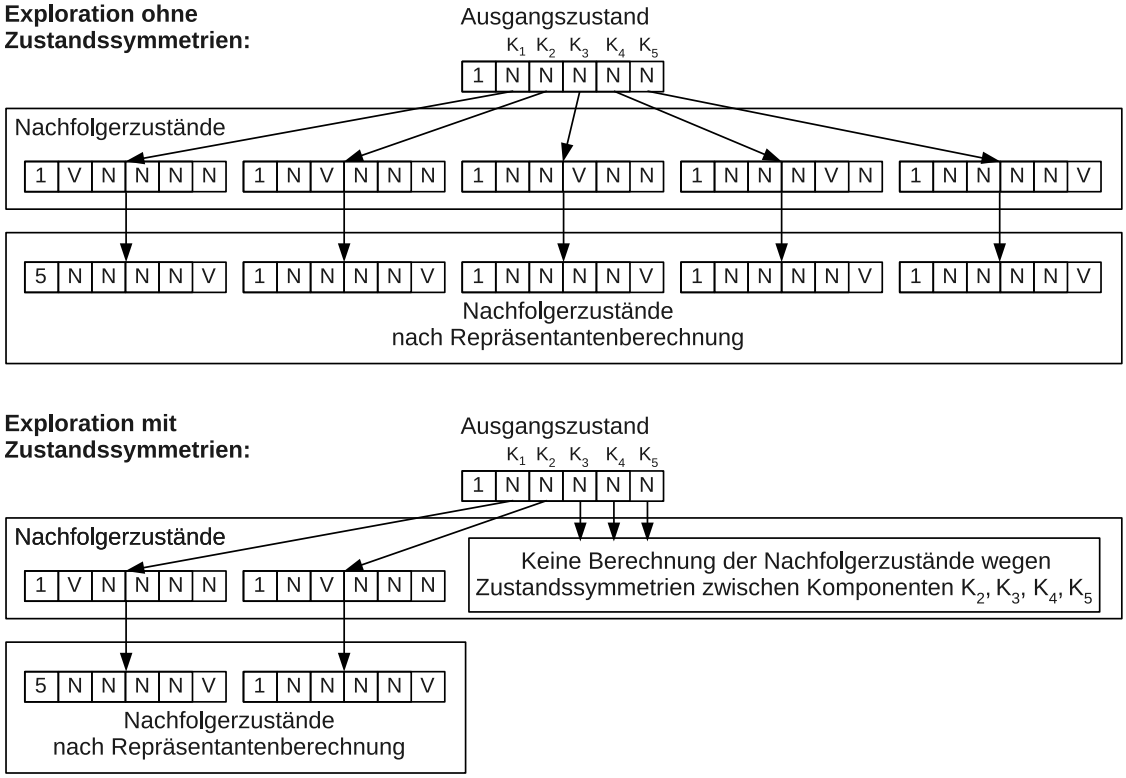

Abbildung 2.10: Beispiel für die Nachfolgerberechnung ohne und mit Ausnutzung von Zustandssymmetrien.

besteht aus zwei Partitionen $P_1 = \{K_1\}$ und $P_2 = \{K_2, K_3, K_4, K_5\}$. Da zur Berechnung aller Repräsentanten von jeder Partition nur eine Komponente ihre Transitionen ausführen muss, explorieren nur Komponente K_1 und K_2 ihre Nachfolgezustände. Damit müssen nur für zwei Nachfolgerzustände Repräsentantenberechnungen durchgeführt werden. Explizite Modellprüfer, für die die Ausnutzung von Zustandssymmetrien implementiert wurde, sind SMC [178] und Murphi [7]. In [178] und [7] sind dazu Ergebnisse von Verifikationsexperimenten zu finden, die besonders für Systeme mit vielen Komponenten deutliche Laufzeitverbesserungen bei Ausnutzung von Zustandssymmetrien aufzeigen. Als wichtig für das Erreichen möglichst großer Laufzeitverbesserungen mit der Ausnutzung von Zustandssymmetrien, hat sich dabei auch die effiziente Berechnung der in einem Zustand vorhandenen Zustandssymmetrien erwiesen.

2.4.3 Modellprüfer mit Symmetrieausnutzung

In diesem Abschnitt werden Erweiterungen der in Abschnitt 2.1.7 vorgestellten Modellprüfer, um Methoden zur zusätzlichen Ausnutzung von Symmetrien beschrieben. Außerdem werden Modellprüfer vorgestellt, die direkt für die Implementierung von Symmetriereduk-

tionsverfahren entwickelt wurden. Neben den in den folgenden Abschnitten aufgeführten Modellprüfern wurden Symmetriereduktionstechniken auch für Modellprüfer für Echtzeitsysteme (z.B. UPPAAL [17, 107]) und für probabilistische Modellprüfer (z.B. PRISM [67, 70, 71, 110, 128, 129]) implementiert.

2.4.3.1 Explizite Modellprüfer mit Symmetrieausnutzung

Beim in Abschnitt 2.1.7.1 beschriebenen expliziten Modellprüfer Murphi wurde die Eingabe von Symmetrien durch einen neu eingeführten Datentyp *Skalarset* (siehe auch Abschnitt 2.4.1.4) ermöglicht ([115, 116]). Durch Ableitung und Benutzung der damit angegebenen Symmetrien kann Murphi direkt eine symmetriereduzierte Quotienten Kripke-Struktur explorieren. Der Modellprüfer SMC [178] erlaubt die Ausnutzung von Symmetrien durch replizierte Komponenten in Kripke-Strukturen, sowie die Ausnutzung von Zustandssymmetrien. Die Eingabe der vorhandenen Symmetrien erfolgt bei SMC nicht durch die Benutzung des Datentyps Skalarset, sondern durch Beschränkungen in der Eingabesprache, die Symmetrien zwischen Komponenten gleichen Typs garantieren. Als Orbit-Repräsentanten werden die ersten Zustände gewählt, die bei der Durchmusterung des Zustandsraums für einen Orbit auftreten. Um das Vergleichen von neu gefundenen Zuständen mit bereits gefundenen Zuständen zu beschleunigen, verwendet SMC einen Approximationsalgorithmus. Dadurch kann es sein, dass nicht die maximal mögliche Speicherersparnis erreicht wird und mehrere Repräsentanten für einen Orbit gespeichert werden.

Erweiterungen des in Abschnitt 2.1.7.1 vorgestellten Modellprüfers SPIN um Symmetriereduktionstechniken wurden in den Paketen SymmSPIN [1, 30] und TopSPIN [65, 66] implementiert. Das Symmetriepaket SymmSPIN wurde zur praktischen Evaluation verschiedener Verfahren zur Repräsentantenberechnung mit dem Modellprüfer SPIN implementiert. Es verwendet wie die Erweiterung des Modellprüfers Murphi den Datentyp Skalarset zur Angabe vorhandener Symmetrien und bietet mehrere Verfahren zur Berechnung von Orbit-Repräsentanten an. TopSPIN verfolgt zur Erkennung vorhandener Symmetrien einen anderen Ansatz. Es analysiert die Kommunikationsstruktur eines eingegebenen Modells automatisch (siehe Abschnitt 2.4.1.4) und leitet daraus eine Symmetriegruppe ab, unter der die zu prüfenden Eigenschaften gültig bleiben. Für TopSPIN wurden ebenfalls mehrere Algorithmen zur Repräsentantenberechnung implementiert. Anhand der gefundenen Symmetriegruppe kann TopSPIN automatisch einen Algorithmus davon auswählen. Durch die Erkennung von Symmetrien durch Analyse der vorhandenen Kommunikationsstruktur, statt der Benutzung des Datentyps Skalarset, ist TopSPIN für eine größere Menge von Symmetriegruppen als SymmSPIN benutzbar.

Für den ebenfalls in Abschnitt 2.1.7.1 bereits vorgestellten Modellprüfer PROB wurde die Ausnutzung von Symmetrien auf zwei verschiedene Arten implementiert. In [135] wurde eine Methode präsentiert, die Permutationen von explorierten Zuständen erzeugt und speichert, um die Modellprüfung zu beschleunigen. Dadurch vergrößert sich zwar der Zustandsraum im Vergleich zur Verwendung der Symmetrien zur Verkleinerung des Zustandsraums, aber es konnten bei Verifikationsexperimenten Laufzeitgewinne erzielt werden. Die zweite für PROB implementierte Möglichkeit zur Ausnutzung von Symmetrien ist der klassische Ansatz der Ausnutzung von Symmetrien zur Verkleinerung des zu

durchmusternden Zustandsraums. Dafür wurden verschiedene Strategien implementiert [137, 136, 182, 186].

2.4.3.2 Symbolische Modellprüfer mit Symmetrieausnutzung

Ein Modellprüfer für die symbolische Modellprüfung mit BDDs mit Ausnutzung von Symmetrien, der den mehrere Repräsentanten-Ansatz (siehe Abschnitt 2.4.2.2) benutzt, ist der Modellprüfer SYMM [57]. Symmetrien können bei ihm durch den Benutzer eingegeben werden. Seine Eingabesprache erlaubt die Spezifikation von Systemen mit mehreren Komponenten, die über globale Variablen kommunizieren. Als Eigenschaften können CTL-Eigenschaften angegeben werden.

Für den in Abschnitt 2.1.7.2 bereits beschriebenen Modellprüfer BOOM wurde zur Ausnutzung von Symmetrien durch replizierte Komponenten das Verfahren der Zählerabstraktion implementiert. Um das Problem der Explosion der lokalen Zustände bei der Zählerabstraktion abzumildern, wurde ein in [13] vorgestellter Ansatz entwickelt. Ebenfalls wurden für den Modellprüfer RuleBase (siehe Abschnitt 2.1.7.2) Erweiterungen zur Ausnutzung von Symmetrien implementiert ([9, 10]). Um vorhandene Symmetrien ausnutzen zu können, müssen diese vom Benutzer eingegeben werden. Die Symmetriereduktion ist dann mit dem in Abschnitt 2.4.2.2 beschriebenen Verfahren zur On-the-fly Repräsentantenauswahl möglich.

2.4.3.3 Der Modellprüfer Sviss

Die in der vorliegenden Arbeit präsentierten Ergebnisse von Verifikationsexperimenten wurden mit einer erweiterten Version des Modellprüfers SVISS [190, 191] durchgeführt. SVISS ist ein BDD-basierter symbolischer Modellprüfer, der das BDD-Paket CUDD [181] benutzt. Für ihn wurden mehrere Ansätze zur Symmetriereduktion implementiert. Symmetriereduktion kann bei SVISS mit dynamischer Symmetriereduktion [76], unter Benutzung der Orbit-Relation und mit dem mehrere Repräsentanten-Ansatz durchgeführt werden (siehe Abschnitte 2.4.2.2 und 2.4.2.3 für Informationen zu den Symmetriereduktionsverfahren). Als Symmetriearten können vollständige Symmetrie und Rotationssymmetrie für Gruppen von Komponenten angegeben werden. Außerdem können Verifikationsläufe ohne Nutzung von Symmetriereduktionstechniken durchgeführt werden. Um Systemmodelle eingeben zu können, bietet SVISS eine C++-Bibliothek zur Modellierung von Systemen mit BDDs an. SVISS kann damit zum Beispiel zur Verifikation von Kommunikationsprotokollen, nebenläufiger Software oder Hardwarekomponenten benutzt werden. Er kann zur Überprüfung von CTL-Eigenschaften verwendet werden, besitzt aber auch spezielle Algorithmen zur Berechnung der Gültigkeit von Invarianten. Aus einem Eingabemodell wird bei SVISS durch Kompilieren ein ausführbarer Verifizierer erzeugt. Bei dessen Ausführung wird die Modellprüfung mit den eingegebenen Eigenschaften durchgeführt. Beim Aufruf zur Ausführung können noch bestimmte Parameter für die Verifikation, wie zum Beispiel Anzahlen von in einem System vorhandenen replizierten Komponenten, angegeben werden.

2.5 BDD-basierte Zustandsraumexploration

In Abschnitt 2.1.5 wurde beschrieben, wie Mengen von Zuständen und Mengen von Transitionen von Kripke-Strukturen durch Boolesche charakteristische Funktionen repräsentiert werden können. Bei der BDD-basierten Modellprüfung werden solche charakteristischen Funktionen symbolisch durch BDDs repräsentiert. Die in der vorliegenden Arbeit präsentierten Neuentwicklungen und die durchgeführten Verifikationsexperimente werden für die Vorwärtserreichbarkeitsanalyse präsentiert, bei der wiederholt Nachfolgerberechnungen durchgeführt werden. Daher werden die nachfolgenden Ausführungen für Nachfolgerberechnungen angegeben. Transitionen können mit BDDs auf verschiedene Arten abgespeichert werden:

1. Verwendung von BDDs zur Repräsentation von Transitionsfunktionen. Für jede Boolesche Variable für Nachfolgerzustände wird deren Transitionsfunktion, die den Wert der Booleschen Variable nach Zustandsübergängen kodiert, durch einen BDD dargestellt.

2. Verwendung von BDDs zur Repräsentation der Transitionsrelation. Die Transitionsrelation enthält alle Paare von aktuellen Zuständen und Nachfolgerzuständen, zwischen denen Zustandsübergänge möglich sind.

In dieser Arbeit wird die zweite Variante verwendet, bei der die Transitionsrelation durch BDDs repräsentiert wird. Daher werden in diesem Abschnitt nur Techniken zur Modellprüfung bei Verwendung von Transitionsrelationen genauer betrachtet. Verfahren zur Nachfolgerberechnung mit Transitionsfunktionen und weitere Referenzen auf Artikel dazu finden sich zum Beispiel in [58, 59, 61, 144, 180]. Transitionsfunktionen sind von ihrer Konstruktion nicht natürlich für die Verifikation von Systemen mit Nichtdeterminismus, wie zum Beispiel die in dieser Arbeit betrachteten asynchronen nebenläufigen Systeme, geschaffen [39]. Die Autoren von [39] vermuten, dass die Verwendung partitionierter Transitionsrelationen im Vergleich mit monolithischen Transitionsrelationen und Transitionsfunktionen häufig am effizientesten ist.

2.5.1 Berechnung von Nachfolgerzuständen mit BDDs

Eine zentrale Berechnung bei der BDD-basierten Modellprüfung ist die Berechnung von aus Zustandsmengen erreichbaren Mengen von unmittelbaren Vorgänger- bzw. Nachfolgerzuständen. Die Berechnung einer Menge unmittelbarer Nachfolgerzustände einer Zustandsmenge kann durch das relationale Produkt (engl. relational product) [8, 168] für die Nachfolgerberechnung beschrieben werden.

Definition 2.35. (Relationales Produkt für die Nachfolgerberechnung) *Seien* $X = \{x_1, x_2, ..., x_n\}$ *und* $X' = \{x'_1, x'_2, ..., x'_n\}$ *Mengen von Booleschen Variablen,* $\vec{x} = (x_1, x_2, ..., x_n)$ *bzw.* $\vec{x}' = (x'_1, x'_2, ..., x'_n)$ *Vektoren über diesen Booleschen Variablen und* $R(\vec{x}, \vec{x}')$ *und* $S(\vec{x}')$ *Boolesche charakteristische Funktionen. Dann heißt der Ausdruck*

$$\exists \vec{x}[R(\vec{x}, \vec{x}') \wedge S(\vec{x})]\{\vec{x}' \leftarrow \vec{x}\} \tag{2.11}$$

relationales Produkt für die Nachfolgerberechnung.

Im relationalen Produkt für die Nachfolgerberechnung ist $R(\vec{x}, \vec{x}')$ die charakteristische Funktion einer Menge von Transitionen und $S(\vec{x})$ ist die charakteristische Funktion der Zustandsmenge, für die Nachfolgerzustände zu berechnen sind. In der durch die charakteristische Funktion R repräsentierten Menge von Transitionen sind die durch diese Transitionen gegebenen Zuordnungen von Zuständen, für die Nachfolgerzustände zu berechnen sind (repräsentiert über Variablen aus X), zu Nachfolgerzuständen (repräsentiert über Variablen aus X') abgespeichert. Die Operation $\exists \vec{x}$ entspricht der existentiellen Quantifizierung (siehe Definition 2.38) bezüglich der Variablen in $\vec{x}$ und $\{\vec{x}' \leftarrow \vec{x}\}$ ist eine Umbenennungsoperation, mit der die Variablen aus $\vec{x}'$ in deren korrespondierende Variablen aus $\vec{x}$ umbenannt werden. Eine Menge von Nachfolgerzuständen $S'(\vec{x})$, die ebenfalls über Variablen der Menge X repräsentiert wird, kann damit mit dem relationalen Produkt für die Nachfolgerberechnung durch

$$S'(\vec{x}) = \exists \vec{x}[R(\vec{x}, \vec{x}') \wedge S(\vec{x})]\{\vec{x}' \leftarrow \vec{x}\}$$

berechnet werden. Für die Nachfolgerberechnung und die Variablenmengen X und X' werden nachfolgend die Begriffe *Variablen für aktuelle Zustände* (in Definition 2.36) und *Variablen für Nachfolgerzustände* (in Definition 2.37) eingeführt.

Definition 2.36. (Variablen für aktuelle Zustände) *Die im relationalen Produkt für die Nachfolgerberechnung* $\exists \vec{x}[R(\vec{x}, \vec{x}') \wedge S(\vec{x})]\{\vec{x}' \leftarrow \vec{x}\}$ *vorkommenden Variablen der Menge* $X = \{x_1, x_2, ..., x_n\}$ *heißen Variablen für aktuelle Zustände.*

Definition 2.37. (Variablen für Nachfolgerzustände) *Die im relationalen Produkt für die Nachfolgerberechnung* $\exists \vec{x}[R(\vec{x}, \vec{x}') \wedge S(\vec{x})]\{\vec{x}' \leftarrow \vec{x}\}$ *vorkommenden Variablen der Menge* $X' = \{x'_1, x'_2, ..., x'_n\}$ *heißen Variablen für Nachfolgerzustände.*

Damit werden aktuelle Zustände über Variablen für aktuelle Zustände der Menge X und unmittelbare Nachfolgerzustände über Variablen für Nachfolgerzustände der Menge X' repräsentiert. Für eine Variable für aktuelle Zustände $x_i \in X$, ist die entsprechende Variable für den unmittelbaren Nachfolgerzustand die Variable $x'_i \in X'$, die den gleichen Index i besitzt. Wie oben erwähnt, entspricht die Operation $\exists \vec{x}$ der existentiellen Quantifizierung bezüglich der Variablen in $\vec{x}$.

Definition 2.38. (**Existentielle Quantifizierung, [144]**) *Sei* $f : \mathbb{B}^n \rightarrow \mathbb{B}$ *eine Boolesche Funktion über der Variablenmenge* $X = \{x_1, x_2, ..., x_n\}$*, dann ist die existentielle Quantifizierung bezüglich der Variablen* x_i *definiert durch:*

$$\exists_{x_i} f = f|_{x_i=1} \vee f|_{x_i=0} \tag{2.12}$$

Die existentielle Quantifizierung besitzt im relationalen Produkt einen höheren Rang als die Umbenennungsoperation. Dadurch ist sie im relationalen Produkt in der vorliegenden Arbeit, wenn keine abweichende besondere Klammerung erfolgt, immer vor der Umbenennungsoperation auszuführen. Die Berechnung des relationalen Produkts kann in einfacher Weise durch drei aufeinander folgende BDD-Traversierungen durchgeführt werden:

1. Die UND-Verknüpfung $(R(\vec{x},\vec{x}')\wedge S(\vec{x}))$ des BDDs für die Transitionsrelation $R(\vec{x},\vec{x}')$ mit dem BDD für die Zustandsmenge $S(\vec{x})$, für die unmittelbare Nachfolgerzustände berechnet werden sollen.

2. Die existentielle Quantifizierung des Ergebnisses der UND-Verknüpfung bezüglich der Variablen für aktuelle Zustände X $(\exists\vec{x}[R(\vec{x},\vec{x}')\wedge S(\vec{x})])$.

3. Das Umbenennen $(\{\vec{x}'\leftarrow\vec{x}\})$ der Variablen für Nachfolgerzustände $(\in X')$ in die entsprechenden Variablen für aktuelle Zustände $(\in X)$ im Ergebnis der existentiellen Quantifizierung $(\exists\vec{x}[R(\vec{x},\vec{x}')\wedge S(\vec{x})]\{\vec{x}'\leftarrow\vec{x}\})$. Der BDD mit dem Ergebnis der existentiellen Quantifizierung besitzt dabei nur noch Knoten für Variablen für Nachfolgerzustände.

Für die UND-Verknüpfung kann der in Abschnitt 2.2.4 beschriebene Algorithmus APPLY benutzt werden. Die existentielle Quantifizierung einer Booleschen Funktion f bezüglich einer Variablen x_i kann für BDDs durch Anwendung von Kofaktor-Operationen und der APPLY-Operation für die ODER-Verknüpfung durchgeführt werden. Zur Berechnung des relationalen Produkts muss die existentielle Quantifizierung bezüglich der Menge der Variablen für aktuelle Zustände X durchgeführt werden. Um dies effizient durchführen zu können, wurden Algorithmen entwickelt, die ebenfalls auf der Kofaktor-Operation und der ODER-Verknüpfung basieren und die die existentielle Quantifizierung bezüglich einer Menge von Variablen in einem BDD-Durchlauf durchführen (siehe [168]). Ebenso wurde eine Modifikation des APPLY-Algorithmus für die UND-Verknüpfung entwickelt, der die Berechnung der UND-Verknüpfung der Transitionsrelation mit der Menge der zu explorierenden Zustände und die existentielle Quantifizierung $(\exists\vec{x}[R(\vec{x},\vec{x}')\wedge S(\vec{x})])$ in einem BDD Durchlauf erledigt [142].

Nach der existentiellen Quantifizierung bezüglich der Variablen für aktuelle Zustände X, besitzt der resultierende BDD bis auf die Terminalknoten nur noch Knoten für Variablen aus X'. Mit Hilfe dieser Knoten werden die explorierten unmittelbaren Nachfolgerzustände über Variablen der Menge X' repräsentiert. Um die Nachfolgerzustände über Variablen der Menge X für aktuelle Zustände zu erhalten, müssen die mit Variablen der Menge X' markierten Knoten noch in Knoten für die diesen entsprechenden Variablen aus der Menge X für aktuelle Zustände umbenannt werden. Das Umbenennen $f\{\vec{x}'\leftarrow\vec{x}\}$ von Booleschen Variablen für Nachfolgerzustände X' in Boolesche Variablen für aktuelle Zustände X kann für einen BDD zu einer Booleschen Funktion f, der nur Knoten für Variablen für Nachfolgerzustände besitzt, durch Umbenennen der Variable jedes Knotens in die entsprechende Variable für aktuelle Zustände erfolgen. Im Folgenden wird in Definition 2.39 zuerst die Umbenennungsoperation für eine Variable eingeführt.

Definition 2.39. (**Umbenennungsoperation für eine Variable**)
Sei $X' = \{x'_1, x'_2, ..., x'_n\}$ eine Menge von Booleschen Variablen, $f : \mathbb{B}^n \to \mathbb{B}$ eine Boolesche Funktion über der Variablenmenge X' und y eine beliebige für die Funktion f nicht wesentliche Boolesche Variable. Dann ist die Umbenennungsoperation $\{x'_i \leftarrow y\}$ zur Umbenennung einer Booleschen Variablen x'_i in die Boolesche Variable y definiert durch:

$$f\{x'_i \leftarrow y\} = (f|_{x'_i=0}) \cdot \overline{y} \vee (f|_{x'_i=1}) \cdot y \tag{2.13}$$

Durch wiederholtes Anwenden von Definition 2.39 kann daraus die im relationalen Produkt für einen Variablenvektor durchzuführende Umbenennungsoperation abgeleitet werden.

Definition 2.40. (**Umbenennungsoperation für einen Variablenvektor**) *Seien $X = \{x_1, x_2, ..., x_n\}$ und $X' = \{x'_1, x'_2, ..., x'_n\}$ disjunkte Mengen von Booleschen Variablen, $\vec{x} = (x_1, x_2, ..., x_n)$ bzw. $\vec{x}' = (x'_1, x'_2, ..., x'_n)$ Vektoren aus diesen Variablen und f eine über den Variablenmengen X und X' definierte Boolesche Funktion, die nur wesentliche Variablen aus der Menge X' besitzt. Dann ist die Umbenennungsoperation $\{\vec{x}' \leftarrow \vec{x}\}$ zur Umbenennung der Variablen aus $\vec{x}'$ in die entsprechenden Variablen mit dem gleichen Index aus $\vec{x}$ definiert durch:*

$$\begin{aligned} f\{\vec{x}' \leftarrow \vec{x}\} =& ((f|_{x'_1=0,\, x'_2=0,\, ...,\, x'_n=0}) \cdot \overline{x_1}\,\overline{x_2} \,...\, \overline{x_n}) \vee ((f|_{x'_1=1,\, x'_2=0,\, ...,\, x'_n=0}) \cdot x_1\overline{x_2} \,...\, \overline{x_n}) \\ & \vee ... \vee ((f|_{x'_1=1,\, x'_2=1,\, ...,\, x'_n=1}) \cdot x_1 x_2 \,...\, x_n) \end{aligned} \tag{2.14}$$

Der Aufwand für die Umbenennung der Variablen der Knoten ist abhängig von der aktuellen Variablenordnung des BDDs und kann exponentiellen Zeitbedarf erfordern [8]. Allerdings kann die Umbenennung einfach durchgeführt werden, wenn alle korrespondierenden Variablen für aktuelle Zustände $x_i \in X$ und Nachfolgerzustände $x'_i \in X'$ in einer Variablenordnung benachbart sind. Dann kann das Umbenennen einfach durch Durchlaufen eines BDDs und die Durchführung der entsprechenden Umbenennung bei Erreichen von Knoten für Variablen aus X' durchgeführt werden. Bei Verwaltung der BDDs in Form von Shared BDDs ist noch darauf zu achten, dass keine Veränderung von Knoten anderer BDDs erfolgt.

Eine Variablenordnung in der Variablen für aktuelle Zustände und deren korrespondierende Variablen für Nachfolgerzustände benachbart sind, nennt man verschachtelte Variablenordnung (engl. interleaved variable ordering) (siehe Abschnitt 2.5.3). Verschachtelte Variablenordnungen erlauben die einfache Umbenennung von Variablen für Nachfolgerzustände in deren korrespondierende Variablen für aktuelle Zustände und umgekehrt. Außerdem bieten solche Variablenordnungen noch weitere Vorteile bei der Repräsentation von Kripke-Strukturen, z.B. die Einhaltung bestimmter Größenschranken bei BDDs für Transitionsrelationen bestimmter Systemtypen (siehe [8]). Aufgrund dieser Vorteile werden sie häufig bei der Modellprüfung verwendet. Ein Algorithmus, der das relationale Produkt

mit BDDs und die darin enthaltenen Operationen für verschachtelte Variablenordnungen auf einmal berechnet wird in Abschnitt 3.1 vorgestellt. Die für diesen Algorithmus in Abschnitt 8.5.1.1 aufgeführten experimentellen Ergebnisse zeigen, dass mit ihm gegenüber der sequentiellen Berechnung des relationalen Produkts große Laufzeitverbesserungen und auch große Verringerungen des Speicherbedarfs erzielt werden können.

2.5.2 Partitionierte Transitionsrelationen

In Abschnitt 2.5.1 wurde die Berechnung von unmittelbaren Nachfolgerzuständen $S'(x)$ einer Zustandsmenge $S(\vec{x})$ bei Benutzung des relationalen Produkts unter Verwendung einer monolithischen Transitionsrelation $R(\vec{x}, \vec{x}')$ beschrieben. Der BDD für eine monolithische Transitionsrelation kann sehr groß sein. Dies ist oft einer der Hauptgründe für hohen Speicherbedarf bei der symbolischen Modellprüfung. Deshalb wurden verschiedene Ansätze entwickelt, um den Speicherbedarf zum Abspeichern der Transitionsrelation zu reduzieren und die Modellprüfung von Systemen mit intraktabel großem monolithischen BDD für die Transitionsrelation zu ermöglichen. Ein Ansatz dazu ist die Benutzung von partitionierten Transitionsrelationen [36, 37]. Bei partitionierten Transitionsrelationen wird die Transitionsrelation in mehrere Teile aufgespalten. Die generierten Teile können dabei oft durch kleine BDDs dargestellt werden und der Speicherbedarf für die Transitionsrelation wird häufig reduziert. In der vorliegenden Arbeit werden asynchrone nebenläufige Systeme betrachtet. Daher werden partitionierte Transitionsrelationen im Folgenden für asynchrone Systeme eingeführt. Die Transitionsrelation solcher Systeme kann durch konjunktiv und disjunktiv partitionierte Transitionsrelationen repräsentiert werden. Bei konjunktiv partitionierten Transitionsrelationen werden die einzelnen BDDs für Teile der Transitionsrelation mit impliziter UND-Verknüpfung und bei disjunktiv partitionierten Transitionsrelationen mit impliziter ODER-Verknüpfung abgespeichert. Disjunktiv partitionierte Transitionsrelationen, die in dieser Arbeit bei Verwendung partitionierter Transitionsrelationen benutzt werden, können für asynchrone Systeme leicht konstruiert werden [37]. Außerdem besitzen sie einige Vorteile gegenüber konjunktiv partitionierten Transitionsrelationen [11].

Definition 2.41. (**Disjunktiv partitionierte Transitionsrelation**) *Sei R ein BDD, der die Transitionsrelation eines asynchronen nebenläufigen Systems mit n Transitionen repräsentiert. Außerdem sei $T_1, T_2, ..., T_m$, mit $m \geq 2$, eine Partition der Menge $\{t_1, t_2, ..., t_n\}$ der Transitionen des Systems. Eine disjunktiv partitionierte Transitionsrelation ist eine Menge $\{R_1, R_2, ..., R_m\}$ von BDDs, mit R_i repräsentiert die Transitionen von Partition T_i und es gilt*

$$R = R_1 \vee R_2 \vee ... \vee R_m$$

Unter Benutzung einer disjunktiv partitionierten Transitionsrelation aus m BDDs kann die Berechnung einer Menge von Nachfolgerzuständen $S'(\vec{x})$ mit dem in Definition 2.35 eingeführten relationalen Produkt durch folgende Gleichung beschrieben werden:

$$S'(\vec{x}) = \exists \vec{x}[(R_1(\vec{x}, \vec{x}') \vee R_2(\vec{x}, \vec{x}') \vee ... \vee R_m(\vec{x}, \vec{x}')) \wedge S(\vec{x})]\{\vec{x}' \leftarrow \vec{x}\}$$

Nachfolgend wird im Beweis von Satz 2.43 gezeigt, dass die Berechnung von Nachfolgerzuständen durch Berechnung des relationalen Produkts für jede Partition einer disjunktiv partitionierten Transitionsrelation durchgeführt werden kann. Dabei wird benutzt, dass für die Operation der existentiellen Quantifizierung $\exists_{x_i} f$ einer Booleschen Funktion f bezüglich einer Variablen x_i (siehe Definition 2.38) die Distributivität bezüglich der Disjunktion ($\vee$) gilt [144]. Im Beweis zu Satz 2.43 wird ebenfalls die Distributivität der Umbenennungsoperation hinsichtlich der Disjunktion ($\vee$) benutzt. Die Distributivität der Umbenennungsoperation wird für die Disjunktion ($\vee$) und für die Konjunktion ($\wedge$) in Lemma 2.42 allgemein für charakteristische Funktionen, bei denen Variablen in die umbenannt werden soll keine wesentlichen Variablen (siehe Definition 2.24) sind, bewiesen. Wie in Abschnitt 2.5.1 bereits erwähnt, besitzt die existentielle Quantifizierung in der vorliegenden Arbeit im relationalen Produkt einen höheren Rang als die Umbenennungsoperation. Aus diesem Grund wird die existentielle Quantifizierung hier im relationalen Produkt vor der Umbenennungsoperation angewandt und die charakteristische Funktion des Resultats von Anwendungen der existentiellen Quantifizierung im relationalen Produkt besitzt keine wesentlichen Variablen aus der Menge der Variablen, in die umbenannt werden soll. Daher kann Lemma 2.42 zum Beispiel für Satz 2.43 verwendet werden. Im Beweis zum nachfolgenden Lemma 2.42 wird die in Definition 2.40 für die Umbenennungsoperation für einen Variablenvektor eingeführte Gleichung 2.14 verwendet.

Lemma 2.42. (Distributivität der Umbenennungsoperation bezüglich Disjunktion und Konjunktion) *Seien $X = \{x_1, x_2, ..., x_n\}$ und $X' = \{x'_1, x'_2, ..., x'_n\}$ zwei Mengen von Booleschen Variablen und $\vec{x} = (x_1, x_2, ..., x_n)$ bzw. $\vec{x}' = (x'_1, x'_2, ..., x'_n)$ Vektoren aus diesen Variablen. Für charakteristische Boolesche Funktionen $A(\vec{x}'), B(\vec{x}')$ über der Variablenmenge X' ist $\{\vec{x}' \leftarrow \vec{x}\}$, die Operation zur Umbenennung der Variablen aus X' in die diesen entsprechenden Variablen aus X, distributiv bezüglich der Disjunktion ($\vee$) und der Konjunktion ($\wedge$) und es gilt*

$$(A(\vec{x}') \vee B(\vec{x}'))\{\vec{x}' \leftarrow \vec{x}\} = A(\vec{x}')\{\vec{x}' \leftarrow \vec{x}\} \vee B(\vec{x}')\{\vec{x}' \leftarrow \vec{x}\}$$

und

$$(A(\vec{x}') \wedge B(\vec{x}'))\{\vec{x}' \leftarrow \vec{x}\} = A(\vec{x}')\{\vec{x}' \leftarrow \vec{x}\} \wedge B(\vec{x}')\{\vec{x}' \leftarrow \vec{x}\}$$

Beweis. Der erste Teil des Beweises wird für die Disjunktion und die Konjunktion gemeinsam durchgeführt. Dazu sei die zweistellige Operation $\circ$ im Folgenden entweder die Disjunktion oder die Konjunktion ($\circ = \vee$ oder $\circ = \wedge$), womit gilt:

$$
\begin{aligned}
&A(\vec{x}')\{x' \leftarrow x\} \circ B(\vec{x}')\{x' \leftarrow x\} \\
&\overset{Def.}{=}_{2.40}(((A(\vec{x}')|_{x'_1=0,\,x'_2=0,\,\ldots,\,x'_n=0}) \cdot \overline{x_1}\,\overline{x_2} \ldots \overline{x_n}) \vee ((A(\vec{x}')|_{x'_1=1,\,x'_2=0,\,\ldots,\,x'_n=0}) \cdot x_1\overline{x_2} \ldots \overline{x_n}) \vee \ \ldots \\
&\qquad \vee ((A(\vec{x}')|_{x'_1=1,\,x'_2=1,\,\ldots,\,x'_n=1}) \cdot x_1 x_2 \ldots x_n)) \circ (((B(\vec{x}')|_{x'_1=0,\,x'_2=0,\,\ldots,\,x'_n=0}) \cdot \overline{x_1}\,\overline{x_2} \ldots \overline{x_n}) \vee \\
&\qquad ((B(\vec{x}')|_{x'_1=1,\,x'_2=0,\,\ldots,\,x'_n=0}) \cdot x_1\overline{x_2} \ldots \overline{x_n}) \vee \ \ldots \ \vee ((B(\vec{x}')|_{x'_1=1,\,x'_2=1,\,\ldots,\,x'_n=1}) \cdot x_1 x_2 \ldots x_n)) \\
&= (((A(\vec{x}')|_{x'_1=0,\,x'_2=0,\,\ldots,\,x'_n=0}) \cdot \overline{x_1}\,\overline{x_2} \ldots \overline{x_n}) \circ ((B(\vec{x}')|_{x'_1=0,\,x'_2=0,\,\ldots,\,x'_n=0}) \cdot \overline{x_1}\,\overline{x_2} \ldots \overline{x_n})) \vee \\
&\quad (((A(\vec{x}')|_{x'_1=0,\,x'_2=0,\,\ldots,\,x'_n=0}) \cdot \overline{x_1}\,\overline{x_2} \ldots \overline{x_n}) \circ ((B(\vec{x}')|_{x'_1=1,\,x'_2=0,\,\ldots,\,x'_n=0}) \cdot x_1\,\overline{x_2} \ldots \overline{x_n})) \vee \ \ldots \\
&\quad \vee (((A(\vec{x}')|_{x'_1=0,\,x'_2=0,\,\ldots,\,x'_n=0}) \cdot \overline{x_1}\,\overline{x_2} \ldots \overline{x_n}) \circ ((B(\vec{x}')|_{x'_1=1,\,x'_2=1,\,\ldots,\,x'_n=1}) \cdot x_1\,x_2 \ldots x_n)) \vee \ldots \\
&\quad \vee (((A(\vec{x}')|_{x'_1=1,\,x'_2=1,\,\ldots,\,x'_n=1}) \cdot x_1 x_2 \ldots x_n) \circ ((B(\vec{x}')|_{x'_1=0,\,x'_2=0,\,\ldots,\,x'_n=0}) \cdot \overline{x_1}\,\overline{x_2} \ldots \overline{x_n})) \vee \\
&\quad (((A(\vec{x}')|_{x'_1=1,\,x'_2=1,\,\ldots,\,x'_n=1}) \cdot x_1 x_2 \ldots x_n) \circ ((B(\vec{x}')|_{x'_1=1,\,x'_2=0,\,\ldots,\,x'_n=0}) \cdot x_1\,\overline{x_2} \ldots \overline{x_n})) \vee \ \ldots \\
&\quad \vee (((A(\vec{x}')|_{x'_1=1,\,x'_2=1,\,\ldots,\,x'_n=1}) \cdot x_1 x_2 \ldots x_n) \circ ((B(\vec{x}')|_{x'_1=1,\,x'_2=1,\,\ldots,\,x'_n=1}) \cdot x_1\,x_2 \ldots x_n))
\end{aligned}
$$

Nun wird der Beweis für die Disjunktion und die Konjunktion separat weitergeführt.

1. Disjunktion ($\circ = \vee$):

$$
\begin{aligned}
&(((A(\vec{x}')|_{x'_1=0,\,x'_2=0,\,\ldots,\,x'_n=0}) \cdot \overline{x_1}\,\overline{x_2} \ldots \overline{x_n}) \vee ((B(\vec{x}')|_{x'_1=0,\,x'_2=0,\,\ldots,\,x'_n=0}) \cdot \overline{x_1}\,\overline{x_2} \ldots \overline{x_n})) \vee \\
&(((A(\vec{x}')|_{x'_1=0,\,x'_2=0,\,\ldots,\,x'_n=0}) \cdot \overline{x_1}\,\overline{x_2} \ldots \overline{x_n}) \vee ((B(\vec{x}')|_{x'_1=1,\,x'_2=0,\,\ldots,\,x'_n=0}) \cdot x_1\,\overline{x_2} \ldots \overline{x_n})) \vee \ldots \\
&\vee (((A(\vec{x}')|_{x'_1=0,\,x'_2=0,\,\ldots,\,x'_n=0}) \cdot \overline{x_1}\,\overline{x_2} \ldots \overline{x_n}) \vee ((B(\vec{x}')|_{x'_1=1,\,x'_2=1,\,\ldots,\,x'_n=1}) \cdot x_1\,x_2 \ldots x_n)) \vee \\
&\ldots \vee (((A(\vec{x}')|_{x'_1=1,\,x'_2=1,\,\ldots,\,x'_n=1}) \cdot x_1 x_2 \ldots x_n) \vee ((B(\vec{x}')|_{x'_1=0,\,x'_2=0,\,\ldots,\,x'_n=0}) \cdot \overline{x_1}\,\overline{x_2} \ldots \overline{x_n})) \\
&\vee (((A(\vec{x}')|_{x'_1=1,\,x'_2=1,\,\ldots,\,x'_n=1}) \cdot x_1 x_2 \ldots x_n) \vee ((B(\vec{x}')|_{x'_1=1,\,x'_2=0,\,\ldots,\,x'_n=0}) \cdot x_1\,\overline{x_2} \ldots \overline{x_n})) \vee \\
&\ldots \vee (((A(\vec{x}')|_{x'_1=1,\,x'_2=1,\,\ldots,\,x'_n=1}) \cdot x_1 x_2 \ldots x_n) \vee ((B(\vec{x}')|_{x'_1=1,\,x'_2=1,\,\ldots,\,x'_n=1}) \cdot x_1\,x_2 \ldots x_n)) \\
&= ((A(\vec{x}')|_{x'_1=0,\,x'_2=0,\,\ldots,\,x'_n=0}) \cdot \overline{x_1}\,\overline{x_2} \ldots \overline{x_n}) \vee ((A(\vec{x}')|_{x'_1=1,\,x'_2=0,\,\ldots,\,x'_n=0}) \cdot x_1\overline{x_2} \ldots \overline{x_n}) \vee \ldots \vee \\
&\quad ((A(\vec{x}')|_{x'_1=1,\,x'_2=1,\,\ldots,\,x'_n=1}) \cdot x_1 x_2 \ldots x_n) \vee ((B(\vec{x}')|_{x'_1=0,\,x'_2=0,\,\ldots,\,x'_n=0}) \cdot \overline{x_1}\,\overline{x_2} \ldots \overline{x_n}) \vee \\
&\quad ((B(\vec{x}')|_{x'_1=1,\,x'_2=0,\,\ldots,\,x'_n=0}) \cdot x_1\overline{x_2} \ldots \overline{x_n}) \vee \ldots \vee ((B(\vec{x}')|_{x'_1=1,\,x'_2=1,\,\ldots,\,x'_n=1}) \cdot x_1\,x_2 \ldots x_n) \\
&= (((A(\vec{x}')|_{x'_1=0,\,x'_2=0,\,\ldots,\,x'_n=0}) \vee (B(\vec{x}')|_{x'_1=0,\,x'_2=0,\,\ldots,\,x'_n=0})) \cdot \overline{x_1}\,\overline{x_2} \ldots \overline{x_n}) \vee \\
&\quad (((A(\vec{x}')|_{x'_1=1,\,x'_2=0,\,\ldots,\,x'_n=0}) \vee (B(\vec{x}')|_{x'_1=1,\,x'_2=0,\,\ldots,\,x'_n=0})) \cdot x_1\overline{x_2} \ldots \overline{x_n}) \vee \ \ldots \vee \\
&\quad (((A(\vec{x}')|_{x'_1=1,\,x'_2=1,\,\ldots,\,x'_n=1}) \vee (B(\vec{x}')|_{x'_1=1,\,x'_2=1,\,\ldots,\,x'_n=1})) \cdot x_1 x_2 \ldots x_n) \\
&= ((A(\vec{x}') \vee B(\vec{x}'))|_{x'_1=0,\,x'_2=0,\,\ldots,\,x'_n=0} \cdot \overline{x_1}\,\overline{x_2} \ldots \overline{x_n}) \vee \\
&\quad ((A(\vec{x}') \vee B(\vec{x}'))|_{x'_1=1,\,x'_2=0,\,\ldots,\,x'_n=0} \cdot x_1\overline{x_2} \ldots \overline{x_n}) \vee \ \ldots \vee \\
&\quad ((A(\vec{x}') \vee B(\vec{x}'))|_{x'_1=1,\,x'_2=1,\,\ldots,\,x'_n=1} \cdot x_1 x_2 \ldots x_n) \\
&= (A(\vec{x}') \vee B(\vec{x}'))\{x' \leftarrow x\}
\end{aligned}
$$

2. Konjunktion ($\circ = \wedge$):

$$
\begin{aligned}
&(((A(\vec{x}'))|_{x'_1=0,\,x'_2=0,\,\ldots,\,x'_n=0}) \cdot \overline{x_1}\,\overline{x_2}\ldots\overline{x_n}) \wedge ((B(\vec{x}'))|_{x'_1=0,\,x'_2=0,\,\ldots,\,x'_n=0}) \cdot \overline{x_1}\,\overline{x_2}\ldots\overline{x_n}))\vee\\
&(((A(\vec{x}'))|_{x'_1=0,\,x'_2=0,\,\ldots,\,x'_n=0}) \cdot \overline{x_1}\,\overline{x_2}\ldots\overline{x_n}) \wedge ((B(\vec{x}'))|_{x'_1=1,\,x'_2=0,\,\ldots,\,x'_n=0}) \cdot x_1\,\overline{x_2}\ldots\overline{x_n})) \vee \ldots\\
&\vee(((A(\vec{x}'))|_{x'_1=0,\,x'_2=0,\,\ldots,\,x'_n=0}) \cdot \overline{x_1}\,\overline{x_2}\ldots\overline{x_n}) \wedge ((B(\vec{x}'))|_{x'_1=1,\,x'_2=1,\,\ldots,\,x'_n=1}) \cdot x_1\,x_2\ldots x_n))\vee\\
&\ldots \vee (((A(\vec{x}'))|_{x'_1=1,\,x'_2=1,\,\ldots,\,x'_n=1}) \cdot x_1x_2\ldots x_n) \wedge ((B(\vec{x}'))|_{x'_1=0,\,x'_2=0,\,\ldots,\,x'_n=0}) \cdot \overline{x_1}\,\overline{x_2}\ldots\overline{x_n}))\\
&\vee(((A(\vec{x}'))|_{x'_1=1,\,x'_2=1,\,\ldots,\,x'_n=1}) \cdot x_1x_2\ldots x_n) \wedge ((B(\vec{x}'))|_{x'_1=1,\,x'_2=0,\,\ldots,\,x'_n=0}) \cdot x_1\,\overline{x_2}\ldots\overline{x_n}))\vee\\
&\ldots \vee (((A(\vec{x}'))|_{x'_1=1,\,x'_2=1,\,\ldots,\,x'_n=1}) \cdot x_1x_2\ldots x_n) \wedge ((B(\vec{x}'))|_{x'_1=1,\,x'_2=1,\,\ldots,\,x'_n=1}) \cdot x_1\,x_2\ldots x_n))\\
&=(((A(\vec{x}'))|_{x'_1=0,\,x'_2=0,\,\ldots,\,x'_n=0}) \cdot \overline{x_1}\,\overline{x_2}\ldots\overline{x_n}) \wedge ((B(\vec{x}'))|_{x'_1=0,\,x'_2=0,\,\ldots,\,x'_n=0}) \cdot \overline{x_1}\,\overline{x_2}\ldots\overline{x_n}))\vee\\
&\quad(((A(\vec{x}'))|_{x'_1=1,\,x'_2=0,\,\ldots,\,x'_n=0}) \cdot x_1\overline{x_2}\ldots\overline{x_n}) \wedge ((B(\vec{x}'))|_{x'_1=1,\,x'_2=0,\,\ldots,\,x'_n=0}) \cdot x_1\,\overline{x_2}\ldots\overline{x_n}))\vee\\
&\quad\ldots \vee (((A(\vec{x}'))|_{x'_1=1,\,x'_2=1,\,\ldots,\,x'_n=1}) \cdot x_1x_2\ldots x_n) \wedge ((B(\vec{x}'))|_{x'_1=1,\,x'_2=1,\,\ldots,\,x'_n=1}) \cdot x_1x_2\ldots x_n))\\
&=(((A(\vec{x}'))|_{x'_1=0,\,x'_2=0,\,\ldots,\,x'_n=0}) \wedge (B(\vec{x}'))|_{x'_1=0,\,x'_2=0,\,\ldots,\,x'_n=0})) \cdot \overline{x_1}\,\overline{x_2}\ldots\overline{x_n})\vee\\
&\quad(((A(\vec{x}'))|_{x'_1=1,\,x'_2=0,\,\ldots,\,x'_n=0}) \wedge (B(\vec{x}'))|_{x'_1=1,\,x'_2=0,\,\ldots,\,x'_n=0})) \cdot x_1\,\overline{x_2}\ldots\overline{x_n}) \vee\ \ldots \vee\\
&\quad(((A(\vec{x}'))|_{x'_1=1,\,x'_2=1,\,\ldots,\,x'_n=1}) \wedge (B(\vec{x}'))|_{x'_1=1,\,x'_2=1,\,\ldots,\,x'_n=1})) \cdot x_1x_2\ldots x_n)\\
&=((A(\vec{x}') \wedge B(\vec{x}'))|_{x'_1=0,\,x'_2=0,\,\ldots,\,x'_n=0} \cdot \overline{x_1}\,\overline{x_2}\ldots\overline{x_n})\vee\\
&\quad((A(\vec{x}') \wedge B(\vec{x}'))|_{x'_1=1,\,x'_2=0,\,\ldots,\,x'_n=0} \cdot x_1\,\overline{x_2}\ldots\overline{x_n}) \vee\ \ldots \vee\\
&\quad((A(\vec{x}') \wedge B(\vec{x}'))|_{x'_1=1,\,x'_2=1,\,\ldots,\,x'_n=1} \cdot x_1x_2\ldots x_n)\\
&=(A(\vec{x}') \wedge B(\vec{x}'))\{x' \leftarrow x\}
\end{aligned}
$$

□

Satz 2.43. (Nachfolgerberechnung für disjunktiv partitionierte Transitionsrelationen) *Die Berechnung einer Menge von Nachfolgerzuständen $S'(\vec{x})$ kann für disjunktiv partitionierte Transitionsrelationen aus m Partitionen durch*

$$
\begin{aligned}
S'(\vec{x}) = [\exists\vec{x}(R_1(\vec{x},\vec{x}')\wedge S(\vec{x}))]\{\vec{x}' \leftarrow \vec{x}\} \vee [\exists\vec{x}(R_2(\vec{x},\vec{x}') \wedge S(\vec{x}))]\{\vec{x}' \leftarrow \vec{x}\} \vee \ldots\\
\vee [\exists\vec{x}(R_m(\vec{x},\vec{x}') \wedge S(\vec{x}))]\{\vec{x}' \leftarrow \vec{x}\}
\end{aligned}
$$

beschrieben werden.

Beweis.

$$
\begin{aligned}
S'(\vec{x}) &= \exists\vec{x}[(R_1(\vec{x},\vec{x}') \vee R_2(\vec{x},\vec{x}') \vee \ldots \vee R_m(\vec{x},\vec{x}')) \wedge S(\vec{x})]\{\vec{x}' \leftarrow \vec{x}\}\\
&= \exists\vec{x}[(R_1(\vec{x},\vec{x}') \wedge S(\vec{x})) \vee (R_2(\vec{x},\vec{x}') \wedge S(\vec{x})) \vee \ldots \vee (R_m(\vec{x},\vec{x}') \wedge S(\vec{x}))]\{\vec{x}' \leftarrow \vec{x}\}\\
&= [\exists\vec{x}(R_1(\vec{x},\vec{x}') \wedge S(\vec{x})) \vee \exists\vec{x}(R_2(\vec{x},\vec{x}') \wedge S(\vec{x})) \vee \ldots \vee \exists\vec{x}(R_m(\vec{x},\vec{x}') \wedge S(\vec{x}))]\{\vec{x}' \leftarrow \vec{x}\}\\
&\overset{\text{Lemma}}{=}_{2.42} [\exists\vec{x}(R_1(\vec{x},\vec{x}') \wedge S(\vec{x}))]\{\vec{x}' \leftarrow \vec{x}\} \vee [\exists\vec{x}(R_2(\vec{x},\vec{x}') \wedge S(\vec{x}))]\{\vec{x}' \leftarrow \vec{x}\} \vee \ldots\\
&\qquad \vee [\exists\vec{x}(R_m(\vec{x},\vec{x}') \wedge S(\vec{x}))]\{\vec{x}' \leftarrow \vec{x}\}
\end{aligned}
$$

□

Da die existentielle Quantifizierung in der vorliegenden Arbeit im relationalen Produkt einen höheren Rang als die Umbenennungsoperation hat (siehe Abschnitt 2.5.1), wäre die in Satz 2.43 verwendete Klammerung zur jeweiligen Ausführung der existentiellen Quantifizierung vor der Umbenennungsoperation nicht notwendig. Diese Klammerung wurde hier zur besseren Verständlichkeit angegeben. Damit kann die Nachfolgerberechnung durch mehrere Berechnungen des relationalen Produkts, mit häufig relativ kleinen BDDs für Partitionen der Transitionsrelation durchgeführt werden. Der Bau des oft großen BDDs der unreduzierten monolithischen Transitionsrelation kann vermieden werden. Wie auch die in dieser Arbeit präsentierten experimentellen Ergebnisse zeigen, kann dies die erfolgreiche Modellprüfung von Systemen mit intraktabler monolithischer Transitionsrelation ermöglichen. Allerdings kann die Verwendung zu kleiner Partitionen in partitionierten Transitionsrelationen zu Laufzeitverschlechterungen führen, so dass die Verwendung größerer Partitionen sinnvoll sein kann. Ebenso kann die gemeinsame Speicherung von Transitionen, deren BDD-Repräsentationen Ähnlichkeiten aufweisen, in einer Partition der Transitionsrelation zu geringeren Knotenzahlen führen. Daher kann die Auswahl der Partitionen bei Verwendung partitionierter Transitionsrelationen großen Einfluss auf Laufzeit und Speicherbedarf der Modellprüfung haben. Ein Ansatz der Verwendung partitionierter Transitionsrelationen für die Modellprüfung asynchroner nebenläufiger Systeme ist die ausschließliche Speicherung von Transitionen einer Komponente in jeder Partition.

Definition 2.44. (**Komponentenweise partitionierte Transitionsrelation**) *Gegeben sei ein asynchrones nebenläufiges System aus $n > 1$ Komponenten und eine disjunktiv partitionierte Transitionsrelation $R = \{R_1, R_2, ..., R_m\}$ aus $m \geq 2$ BDDs, zur Repräsentation der Transitionsrelation des Systems. Dann heißt die disjunktiv partitionierte Transitionsrelation komponentenweise partitioniert, wenn in jeder Partition R_j ($j \in \{1, ..., m\}$) nur Transitionen einer Komponente i ($i \in \{1, ..., n\}$) gespeichert sind.*

Neben der Auswahl der Partitionen ist für die Optimierung der Effizienz der Modellprüfung auch die Reihenfolge, in der Partitionen für Bildberechnungen benutzt werden, wichtig. Für beide Probleme wurden Ansätze zur Optimierung der Verwendung partitionierter Transitionsrelationen entwickelt (siehe z.B. [24, 36, 95, 164, 179, 195]). Verfahren zur effizienten Durchmusterung des Zustandsraums, auch bei Verwendung partitionierter Transitionsrelationen, werden in Abschnitt 2.5.7 präsentiert. Bei Verwendung konjunktiv partitionierter Transitionsrelationen ist zu beachten, dass die existentielle Quantifizierung nicht distributiv gegenüber der Konjunktion ist. Wenn Partitionen einer konjunktiv partitionierten Transitionsrelation jedoch nicht von allen Variablen abhängen, für die die existentielle Quantifizierung durchgeführt werden soll, muss die existentielle Quantifizierung bezüglich solcher Variablen für diese auch nicht durchgeführt werden. Dadurch ist es hier möglich Variablen von denen wenige Partitionen abhängen schon frühzeitig auszuquantifizieren, um damit zu versuchen die BDD Größe während Nachfolgerberechnungen mit weiteren Partitionen möglichst klein zu halten. Dieser Ansatz wird *Early Quantification* [56] genannt.

2.5.3 Identitätsmuster bei verschachtelten Variablenordnungen

In der vorliegenden Arbeit wurden bei Verifikationsexperimenten für BDDs verschachtelte Variablenordnungen verwendet. Bei solchen Variablenordnungen sind Variablen für aktuelle Zustände und deren korrespondierende Variablen für Nachfolgerzustände immer direkt benachbart. Da sie im Vergleich mit anderen Variablenordnungen häufig zu kleineren BDDs für Transitionsrelationen führen und Vorteile für Algorithmen zur Nachfolgerberechnung liefern, sind sie für die Repräsentation von Transitionssystemen zu bevorzugen [8]. Ein Vorteil der Verwendung verschachtelter Variablenordnungen ist die dort effizient mögliche Berechnung der im relationalen Produkt enthaltenen Umbenennungsoperation. Bei verschachtelten Variablenordnungen können Variablen zur Kodierung von Nachfolgerzuständen direkt und einfach in die entsprechenden benachbarten Variablen für aktuelle Zustände umbenannt werden. Für beliebige Variablenordnungen $<_{level}$, in denen sich die relative Ordnung der Variablen für aktuelle Zustände auch von der relativen Ordnung der entsprechenden Variablen für Nachfolgerzustände unterscheiden kann (z.B. $x_i <_{level} x_j$ aber $x'_j <_{level} x'_i$), ist die Umbenennungsoperation aufwendiger und nicht mehr so einfach durchführbar. Es gibt daher keinen Algorithmus, der die Umbenennungsoperation für beliebige Variablenordnungen in linearer Zeit durchführt [8].

BDDs für Transitionsrelationen asynchroner nebenläufiger Systeme mit Transitionslokalität (siehe Abschnitt 2.3) besitzen bei Verwendung verschachtelter Variablenordnungen häufig viele sogenannte Identitätsmuster (engl. identity patterns). Dies ist besonders bei komponentenweise partitionierten Transitionsrelationen, in denen Partitionen nur Transitionen einer Komponente beinhalten, der Fall. In asynchronen nebenläufigen Systemen mit Transitionslokalität hängen Transitionen von Komponenten nur von den Werten der globalen Variablen und von den Werten der eigenen lokalen Variablen ab. Durch Ausführen von Transitionen durch eine Komponente i ändern sich dadurch die lokalen Zustände der anderen Komponenten $j \neq i$ nicht. Die lokalen Zustände im Nachfolgerzustand entsprechen für diese Komponenten hier den lokalen Zuständen im Ausgangszustand. Für dieses Verhalten muss mit der Transitionsrelation sichergestellt werden, dass für die entsprechenden Variablen für Nachfolgerzustände nur die gleichen Belegungen, wie für deren korrespondierende Variablen für aktuelle Zustände erlaubt sind. In der BDD-Repräsentation einer Transitionsrelation wird dieses Verhalten bei verschachtelten Variablenordnungen durch Identitätsmuster repräsentiert. Wird keine verschachtelte Variablenordnung verwendet, müssen bei Transitionen solchen korrespondierenden Variablen ebenfalls die gleichen Werte zugewiesen werden. Wenn ein Variablenpaar korrespondierender Variablen in der Variablenordnung nicht benachbart ist, erfolgt die Repräsentation dieses Verhaltens aber nicht durch Identitätsmuster. Daher wird bei Variablenpaaren aus Variable für aktuelle Zustände und korrespondierender Variable für Nachfolgerzustände, die in einer Variablenordnung nicht benachbart angeordnet sind, von einer Repräsentation dieses Verhaltens durch Identitätstransformationen gesprochen.

Im Folgenden wird das Aussehen von Identitätsmustern beschrieben. Dazu ist ein Beispiel eines Identitätsmusters in einem BDD in Abbildung 2.11 angegeben. In der Abbildung ist x_k eine Boolesche Variable für aktuelle Zustände und x_{k+1} die korrespondierende Variable für Nachfolgerzustände. Der angegebene Ausschnitt eines BDDs wertet nur auf

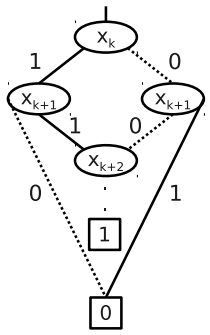

Abbildung 2.11: Beispiel für ein Identitätsmuster in einem BDD.

den Booleschen Wert 1 aus, wenn x_k und x_{k+1} die gleichen Werte zugewiesen bekommen. Ansonsten wertet er auf den Booleschen Wert 0 aus. Wenn in einem BDD Identitätsmuster vorhanden sind, kann der BDD für einen Pfad mit Identitätsmustern daher nur auf den Booleschen Wert 1 auswerten, wenn jeweils beide Booleschen Variablen eines Identitätsmusters die gleichen Werte zugewiesen bekommen. Sind in einem BDD viele Identitätsmuster vorhanden, kann das Abspeichern dieser viel Speicherbedarf benötigen. Außerdem müssen die Knoten von Identitätsmustern von Algorithmen zur Nachfolgerberechnung berücksichtigt werden, was einiges an Laufzeit kosten kann. Deshalb wurden Ansätze entwickelt, bei denen Identitätsmuster in Transitionsrelationen nicht mehr explizit abgespeichert werden müssen.

2.5.4 Transitionsrelationen ohne Identitätsmuster

Das Vermeiden des Abspeicherns von Identitätsmustern in BDDs für Transitionsrelationen kann zu Laufzeit- und Speicherplatzeinsparungen führen. Daher wurden verschiedene Techniken dazu entwickelt. In [39, 56] wird für BDDs variablenweise disjunktiv partitionierter Transitionsrelationen vorgeschlagen, nur Beschränkungen des Werts und die entsprechenden Abhängigkeiten für Wertänderungen der Variable abzuspeichern, für die eine Partition erstellt wurde. Für Variablen deren Werte unverändert übernommen werden sollen, werden keine Identitätsmuster im BDD abgespeichert. Es wird implizit die Interpretation verwendet, dass diese Variablen ihre Werte im aktuellen Zustand in Nachfolgerzuständen beibehalten. Außerdem wird eine modifizierte Formulierung des relationalen Produkts präsentiert, die diese implizite Interpretation berücksichtigt. Ein für die Modellprüfung von Software entwickelter Ansatz wird in [11] präsentiert. Es wird dort eine disjunktiv partitionierte Transitionsrelation aus sogenannten partiellen disjunktiven Partitionen (engl. partial disjunctive partitions) vorgestellt. Partitionen werden dort für den Programmzähler und eine weitere Zustandsvariable aufgebaut. Dabei wird ausgenutzt, dass bei Software in jedem Ausführungsschritt nur wenige Variablen ihre Werte verändern. Zustandsvariablen, für die eine Partition nicht gebaut wurde, behalten ihre Werte bei Ausführung von Transitionen der Partition bei. Für diese werden in partiellen disjunktiven Transitionsrelationen keine Beschränkungen, wie sie in traditionellen disjunk-

tiven Formen vorhanden sind, angegeben. Außerdem wird ein modifizierter Algorithmus skizziert, der die Nachfolgerberechnung mit partiellen disjunktiven Transitionsrelationen ermöglicht. Dieser sorgt dafür, dass für Variablen für die keine Beschränkungen in Partitionen vorhanden sind, die unveränderten Werte der Variablen in Nachfolgerzuständen übernommen werden. Durch die Nutzung von partiellen disjunktiven Transitionsrelationen wird der Speicherbedarf für Transitionsrelationen reduziert. Mit der modifizierten Nachfolgerberechnung verspricht man sich auch Laufzeitverbesserungen.

Eine den Entscheidungsdiagrammen ähnliche Datenstruktur sind die Matrix-Diagramme (MxDs, engl. Matrix Diagrams) [45, 149]. Sie sind gerichtete azyklische Graphen, deren Nichtterminalknoten Matrizen von Zeigern auf andere Knoten sind. In [150] wurde eine Variante von MxDs eingeführt, bei der die Terminalknoten die Booleschen Werte 0 und 1 besitzen und die um Identitätsmuster, die in Transitionsrelationen asynchroner Systeme häufig vorkommen, reduziert sind. In [48] wurde eine Regel zur Entfernung von Identitätsmustern aus Mehrwertigen Entscheidungsdiagrammen (MDDs, engl. Multi-valued Decision Diagrams) [125] eingeführt. MDDs sind den in [150] präsentierten MxDs ähnlich. Anstatt Matrizen von Zeigern als Nichtterminalknoten, sind die Nichtterminalknoten in MDDs Vektoren von Zeigern. Die in [48] präsentierte Regel zur Entfernung von Identitätsmustern, kann die Entfernung einer größeren Menge an Identitätsmustern, als der oben für MxDs beschriebene Ansatz erlauben.

Zwei weitere Ansätze, die ebenfalls keine Identitätsmuster in Transitionsrelationen abspeichern, wurden in [194] und [26] vorgestellt. Sie wurden zur effizienteren Unterstützung der symbolischen Modellprüfung, bei zu Beginn der Durchmusterung des Zustandsraums noch nicht feststehenden Wertebereichen von Zustandsvariablen entworfen. Bei sich vergrößerndem Wertebereich einer Zustandsvariablen muss die Transitionsrelation während der Durchmusterung des Zustandsraums angepasst werden. Dies kann einigen Aufwand erfordern. Ansonsten können Erweiterungen des Wertebereichs von Variablen aufgrund der verwendeten Reduktionsregeln (siehe z.B. [48, 194]) zu Fehlinterpretationen führen, die zu korrigieren sind. Zusätzlich kann das Löschen der Ergebnistabelle (siehe Abschnitt 2.2.3) mit Resultaten bereits ausgeführter Operationen erforderlich sein. Der in [194] präsentierte Ansatz, der auf der Arbeit von [48] mit den eingeführten Regeln zur Entfernung von Identitätsmustern für MDDs basiert, schlägt zur Effizienzsteigerung und Behebung dieser Nachteile die Verwendung von erweiterten Mehrwertigen Entscheidungsdiagrammen (erweiterte MDDs, engl. extended Multi-valued Decision Diagrams) vor. Als Unterschied zu MDDs besitzen Nichtterminalknoten von erweiterten MDDs eine unendliche Anzahl von mit unterschiedlichen natürlichen Zahlen $i \in \mathbb{N}$ markierten Kanten auf Nachfolgerknoten, wobei es eine natürliche Zahl $j \in \mathbb{N}$ gibt, so dass alle Kanten mit Markierung $i > j$ auf den selben Nachfolgerknoten zeigen. Zusätzlich werden in [194] Algorithmen für Operationen mit erweiterten MDDs präsentiert.

In [26] wird ein Algorithmus zur symbolischen Erreichbarkeitsanalyse für die Modellprüfung von Systemen mit Transitionen, die nur von Teilen von Zuständen abhängen, präsentiert. Es werden Gruppen von Transitionen, die jeweils von gleichen Teilen von Zuständen abhängen, zu Partitionen einer partitionierten Transitionsrelation zusammengefasst. Die Gruppen werden dynamisch in jeder Iteration des Algorithmus so angepasst,

dass alle für die Exploration nötigen Transitionen enthalten sind. Die Transitionsrelationen für Gruppen von Transitionen besitzen keine Identitätsmuster für nicht betroffene Zustandsvariablen. Zur Repräsentation der Transitionsrelation werden Listenbasierte Entscheidungsdiagramme (LDDs, engl. List Decision Diagrams) eingeführt und verwendet. LDDs sind MDDs ähnlich. Sie sind gerichtete azyklische Graphen, die Terminalknoten für die Booleschen Werte besitzen. Anstatt Nichtterminalknoten mit mehreren Nachfolgern wie bei MDDs, wird bei LDDs eine verkettete Liste von Nichtterminalknoten mit zwei Nachfolgerknoten verwendet.

2.5.5 Transformationen der Eingabevariablen von BDDs

Bei der Repräsentation mehrerer Boolescher Funktionen durch gemeinsam genutzte BDDs (SROBDDs, siehe Abschnitt 2.2.3), können Gemeinsamkeiten bei der Repräsentation der Booleschen Funktionen ausgenutzt werden. Gemeinsamkeiten zwischen Booleschen Funktionen, die sich nur in den Variablen, über denen sie definiert sind, unterscheiden, lassen sich damit aber häufig nicht ausnutzen. Solche Booleschen Funktionen kommen zum Beispiel bei der Modellierung von Speicherschaltungen vor, bei denen sich die Funktionen für verschiedene Bits eines gespeicherten Wortes nur beim Datenbit unterscheiden und für die die Adressdekodierung gleich ist. Deshalb wurden Ansätze entwickelt, die den Speicherbedarf durch Ausnutzen von bis auf unterschiedliche Variablenmarkierungen isomorpher Teile von BDDs reduzieren.

In [148] werden Kantenattribute verwendet, um das gemeinsame Abspeichern von Teilgraphen, die bis auf um eine Konstante verschobene Variablenindizes isomorph sind, zu ermöglichen. Es werden dazu keine Informationen über Variablenindizes von Knoten gespeichert, sondern Kantenattribute benutzt, um Differenzen zwischen den Variablenindizes aufeinander folgender Knoten anzuzeigen. Für jeden ursprünglichen Teilgraphen gibt es eine Kante zum Wurzelknoten des neuen Teilgraphen. Diese Kante ist mit dem zu verwendenden absoluten Variablenindex des Wurzelknotens markiert, der beim Traversieren von Pfaden auf die auf den Kanten angegebenen Differenzen addiert wird. Eine Differentielle BDDs (ΔBDDs, engl. Differential BDDs) genannte Verallgemeinerung dieses Ansatzes und Transformationen für ΔBDDs wurden in [6] eingeführt. Bei ΔBDDs sind die Knoten mit Integerwerten markiert. Der Integerwert eines Knotens v gibt die Differenz zwischen den Variablenindizes seiner Vorgänger und dem Variablenindex von v an. Der dem Wurzelknoten eines ΔBDDs zugeordnete Integerwert gibt die erste Variable an, von der der ΔBDD abhängt. Der Variablenindex eines Knotens kann damit aus dem Variablenindex des Wurzelknotens und den Integerwerten von Knoten auf Pfaden zum Knoten bestimmt werden. Die in [148] präsentierten zuvor erwähnten BDDs, bei denen Kantenattribute Differenzen zwischen Variablenindizes angeben, können bis auf die Markierung von Knoten mit Integerwerten, statt der Benutzung von Kantenattributen, aus ΔBDDs durch Anwendung von in [6] vorgestellten Transformationen erzeugt werden. ΔBDDs können in Abhängigkeit der zu repräsentierenden Booleschen Funktion zu einer exponentiellen Verringerung der Knotenzahl im Vergleich mit BDDs führen. Im schlimmsten Fall kann es bei ihrer Verwendung anstatt von BDDs einen quadratischen Anstieg der Knotenzahl geben.

Die Verwendung von geordneten Funktionsvorlagen (engl. Ordered Function Templates) wurde in [99] für eine Repräsentation von Booleschen Funktionen durch BDDs vorgestellt. Eine Funktionsvorlage ist eine Repräsentation einer Booleschen Funktion, die nicht fest an eine bestimmte Variablenmenge gebunden ist. Konkrete Boolesche Funktionen werden durch Tupel bestehend aus einer Funktionsvorlage und einer Liste von Variablen repräsentiert. Bei als BDD gespeicherten Funktionsvorlagen erhält man durch Umbenennen der formellen Variablen von Knoten der Funktionsvorlage, in die entsprechenden in der Liste angegebenen Variablen, einen BDD zur als Tupel gespeicherten Funktion. Unter Vernachlässigung der Liste mit den einzusetzenden Variablen, besitzt eine einzelne als BDD gespeicherte Funktionsvorlage die gleiche Größe wie ein BDD für die Funktion. Werden mehrere Funktionsvorlagen in Form von gemeinsam genutzten BDDs gespeichert, können sich allerdings durch Ausnutzen gleicher Teilgraphen große Verringerungen des Speicherbedarfs ergeben. In [99] werden auch einfache Algorithmen für Boolesche Operationen mit Funktionsvorlagen angegeben. Diese bauen aus Funktionsvorlagen zuerst BDDs über den in den Listen angegebenen Variablen auf und führen damit gewöhnliche Algorithmen für Boolesche Operationen aus. Bei den in [99] aufgeführten experimentellen Ergebnissen führte dies im Vergleich mit der Benutzung von BDDs zu deutlichen Laufzeitverschlechterungen. Ein Vorteil der geordneten Funktionsvorlagen ist, dass für jede repräsentierte Funktion andere Umbenennungen der Variablen verwendet werden können. Dies kann das Ausnutzen von Gemeinsamkeiten zwischen Teilgraphen ermöglichen, die mit den ersten beiden in diesem Abschnitt beschriebenen Verfahren nicht ausgenutzt werden können.

Andere Ansätze versuchen möglichst kleine BDDs zu finden, in dem sie die Variablen über denen BDDs definiert sind durch lineare Transformationen dieser Variablen ersetzen [103, 104, 143]. Anstatt, wie bei reinen Verfahren zum Verändern der Variablenordnung, nur die Reihenfolge der Variablen zu verändern, können hier Variablen auch durch lineare Kombinationen von Variablen ersetzt werden.

Auch wurden Entscheidungsdiagramme untersucht, bei denen auf verschiedenen Pfaden unterschiedliche Variablenordnungen erlaubt sind. In ROBDDs (da in dieser Arbeit immer geordnete reduzierte BDDs (ROBDDs) verwendet werden, werden diese außer in Abschnitt 2.2, in dem BDDs eingeführt werden, immer als BDDs bezeichnet) wird auf allen Pfaden von der Wurzel zu den Terminalknoten die gleiche Variablenordnung verwendet und jede Variable darf auf einem Pfad höchstens einmal vorkommen. Freie BDDs (engl. Free BDDs) [97] sind BDDs, die auf verschiedenen Pfaden unterschiedliche Variablenordnungen erlauben und bei denen eine Variable auf einem Pfad ebenfalls höchstens einmal vorkommen darf. In Graphgesteuerten BDDs (engl. Graph-driven BDDs) [177] werden sogenannte Orakel-Graphen verwendet, um die Reihenfolge von Variablen auf Pfaden abzuspeichern. Damit können in Abhängigkeit der Werte von Variablen verschiedene Variablenordnungen verwendet werden. Ein Ansatz zur Wiederverwendung von Variablen bei konjunktiv partitionierten Transitionsrelationen, der Early Quantification (siehe Abschnitt 2.5.2) verwendet, wurde in [198] vorgestellt. In [198] werden ausquantifizierte Variablen wiederverwendet, die bei der Nachfolgerberechnung mit noch zu bearbeitenden Partitionen der Transitionsrelationen nicht mehr benötigt werden. Dies geschieht durch Umbenennung anderer Variablen in diese Variablen. Dadurch existieren größere Möglich-

keiten zur Mehrfachbenutzung existierender Knoten, was den Speicherbedarf reduzieren kann.

2.5.6 Weitere Verfahren zur BDD-basierten Nachfolgerberechnung

Die Berechnung der Nachfolgerzustände einer Zustandsmenge (siehe Abschnitt 2.5.1) ist eine wichtige Operation bei der symbolischen BDD-basierten Modellprüfung. Ein Grund für einen hohen Speicherbedarf bei der Modellprüfung sind häufig dabei auftretende große BDDs für Zwischenergebnisse. Diese können einen zu großen Speicherbedarf beanspruchen, auch wenn das Ergebnis der durchgeführten Berechnung möglicherweise erfolgreich gespeichert werden könnte. Deshalb ist es wichtig Algorithmen und Verfahren einzusetzen, die versuchen die Zwischenergebnisse klein zu halten. In den vorangehenden Abschnitten wurden einige Techniken vorgestellt, die zur effizienten Repräsentation von Transitionsrelationen genutzt werden können. In diesem Abschnitt werden Verfahren zur effizienteren Nachfolgerberechnung mit BDDs beschrieben. Eine Möglichkeit die BDDs während der Zustandsdurchmusterung möglichst klein zu halten, ist die Verwendung von Strategien zum dynamischen Umordnen der Variablenordnung von BDDs (siehe Abschnitt 2.2.2). Statische Verfahren zur Bestimmung einer guten Variablenordnung ermöglichen es zwar gute Variablenordnungen für die ersten Schritte einer Erreichbarkeitsanalyse zu erstellen, es erfolgt hier aber keine Anpassung an Veränderungen der zu repräsentierenden Zustandsmengen während der Durchmusterung des Zustandsraums. Dynamische Verfahren zum Umordnen der Variablenordnung ermöglichen hingegen die Anpassung der Variablenordnung an sich während der Zustandsraumdurchmusterung ändernde Bedingungen. Dies benötigt eine gewisse Rechenzeit, kann aber manchmal zu deutlich kleineren BDDs führen. Dadurch kann möglicherweise auch die Laufzeit von Operationen mit BDDs verringert und die Verifikation von Systemen, die vorher nicht erfolgreich verifiziert werden konnten, ermöglicht werden.

In Abschnitt 2.5.2 wurden partitionierte Transitionsrelationen eingeführt. Wie dort beschrieben kann die Nachfolgerberechnung bei disjunktiv partitionierten Transitionsrelationen separat mit den einzelnen Partitionen durchgeführt werden. Für konjunktiv partitionierte Transitionsrelationen wurden Ansätze entwickelt, die versuchen Variablen so früh wie möglich auszuquantifizieren. Eine weitere Optimierung der Nachfolgerberechnung ist die Verwendung von Algorithmen, die die zur Berechnung des relationalen Produkts nötigen Operationen der Umbenennung von Variablen und der existentiellen Quantifizierung während der Berechnung der im relationalen Produkt enthaltenen UND-Verknüpfung durchführen. Ein Algorithmus, der die Operationen des relationalen Produkts auf einmal berechnet, wird in Abschnitt 3.1 der vorliegenden Arbeit vorgestellt. In [142] wurde ein Algorithmus eingeführt, der die UND-Verknüpfung der BDDs und die existentielle Quantifizierung gemeinsam durchführt. Beide Algorithmen können zu großen Laufzeitverbesserungen und Verringerungen des Speicherbedarfs führen. Operatoren und entsprechende Algorithmen, die statt der im relationalen Produkt enthaltenen UND-Verknüpfung verwendet werden können, wurden in [60, 144, 184] präsentiert. Die vorgestellten Operatoren basieren auf der Bildung verallgemeinerter Kofaktoren [127]. Die Idee dabei ist, die Funktionswerte von Pfaden im BDD des Ergebnisses der Operatoren, die zu Pfaden im BDD der

Zustandsmenge korrespondieren, die auf den Funktionswert 0 auswerten, umzudefinieren. Das Ziel davon ist, dass die neuen Funktionswerte zu kleineren BDDs für Zwischenergebnisse führen. Dies wird so durchgeführt, dass das Ergebnis der Berechnung des relationalen Produkts dabei unverändert bleibt.

Soll die Menge der von den Anfangszuständen erreichbaren Zustände für Systeme mit sehr hoher sequentieller Tiefe (wie z.B. Zählerschaltungen) berechnet werden, können bei der symbolischen BDD-basierten Modellprüfung zum Ermitteln aller erreichbaren Zustände sehr viele Nachfolgerberechnungen nötig sein. Bei Verwendung des relationalen Produkts werden in jeder Nachfolgerberechnung nur unmittelbare Nachfolgerzustände der zu explorierenden Zustandsmenge berechnet. Eine Technik, die für Nachfolgerberechnungen nicht die Transitionsrelation eines Systems nutzt und die die Anzahl der notwendigen Bildberechnungen reduziert ist die des *iterative squaring* [38]. Beim iterative squaring wird der transitive Abschluss der Transitionsrelation berechnet. Der transitive Abschluss enthält alle Paare von Zuständen, die über Folgen von Transitionen durch mindestens einen Pfad verbunden sind. Er kann in Logarithmus der maximalen Distanz zwischen zwei Zuständen vielen Iterationen berechnet werden. Mit einer Nachfolgerberechnung können dann alle von den Anfangszuständen erreichbaren Zielzustände dieser Relation exploriert werden. Ein Nachteil dieses Vorgehens ist, dass die BDDs, die während der Ermittlung des transitiven Abschlusses berechnet werden, sehr groß sein können.

Ein Ansatz, der die Verringerung der Größe von BDDs zur Repräsentation von Zustandsmengen als Ziel hat, wird in [40] präsentiert. Dort werden BDDs zur Repräsentation von Zustandsmengen durch Bildung von Kofaktoren bezüglich einer Variable in nicht überlappende Teile aufgespalten. Es wird versucht die Variable so zu wählen, dass daraus möglichst balancierte BDDs für die entstehenden Teile mit minimaler gesamter BDD-Größe resultieren. Nachfolgerberechnungen können dann separat für generierte Teile von Zustandsmengen durchgeführt werden, was zu kleineren Zwischenergebnissen führen kann. In [153] wurden partitionierte ROBDDs (engl. partitioned ROBDDs) [154] zur effizienteren Repräsentation von Zustandsmengen benutzt. Dort können für BDDs für Partitionen der Zustandsmenge verschiedene Variablenordnungen benutzt werden, was zu einer großen Reduktion des Speicherbedarfs und kleinen Zwischenresultaten führen kann.

2.5.7 Algorithmen zur Zustandsraumdurchmusterung

In Abschnitt 2.1.6.3 wurde ein Algorithmus zur Vorwärtserreichbarkeitsanalyse bei der symbolischen Modellprüfung präsentiert. Bei Verwendung von BDDs zur Repräsentation von Zustandsmengen bzw. der Transitionsrelation, werden in diesem Algorithmus sukzessive BDDs mit erreichbaren Nachfolgerzuständen berechnet, bis alle von den Anfangszuständen erreichbaren Zustände ermittelt wurden. Der Algorithmus wurde für die Verwendung einer monolithischen Transitionsrelation präsentiert. In einer Nachfolgerberechnung werden dort alle in einem Schritt durch Ausführen von Transitionen der Transitionsrelation erreichbaren Nachfolgerzustände berechnet. Das Vorgehen des Algorithmus entspricht einer Breitensuche zur Exploration des Zustandsraums. In jeder Iteration des Algorithmus werden aus der Menge der gerade berechneten Nachfolgerzustände *ExploredStates* die bereits vorher explorierten Zustände, die in der Menge *Reached* angesammelt werden, ent-

fernt. Das Ergebnis ist die Menge der weiter zu explorierenden Zustände *Unexplored*. Dies wird gemacht, um mit möglichst kleinen Zustandsmengen arbeiten zu können, in der Hoffnung dadurch die bei Nachfolgerberechnungen auftretenden BDDs klein halten zu können. Die Menge *ToExplore* wird dann auf die Menge *Unexplored* gesetzt und in der nächsten Iteration werden die möglichen Nachfolgerzustände der neuen Menge *ToExplore* berechnet. Eine kleinere mit einem BDD zu repräsentierende Zustandsmenge bedeutet aber nicht immer einen kleineren BDD. Aufgrund der Reduktionsregeln für BDDs können größere Zustandsmengen zu kleineren BDDs führen. Neben der Menge der neu gefundenen Zustände (*Unexplored*) als weiter zu explorierender Zustandsmenge (*ToExplore*=*Unexplored*, siehe Zeile 6 in Algorithmus 1), lässt sich bei Beibehaltung der Korrektheit des Algorithmus für *ToExplore* auch jede andere Menge *ToExplore* mit

$$Unexplored \subseteq ToExplore \subseteq Unexplored \cup Reached$$

verwenden. Sind in der Menge *ToExplore* Zustände enthalten, die bereits exploriert wurden, werden diese Zustände erneut weiter exploriert. Bei Benutzung von BDDs kann dies aber zu einem kleineren BDD und da die Ausführungszeit vieler Operationen mit BDDs von der Größe der beteiligten BDDs abhängt, auch zu schnelleren Nachfolgerberechnungen führen. Deshalb ist es bei der BDD-basierten Modellprüfung wünschenswert die Menge *ToExplore* so zu wählen, dass sie zu kleineren BDDs führt. In [58] wird vorgeschlagen zur Bestimmung der Menge *ToExplore* die Berechnung eines verallgemeinerten Kofaktors [127] zu benutzen. Diese nutzen die aus der oben angegeben Bedingung für die Wahl von *ToExplore* resultierenden Freiheitsgrade für die Wahl des Funktionswerts von Variablenbelegungen aus, um eine Menge *ToExplore* mit möglichst kleiner BDD Repräsentation zu erhalten.

Mit der Absicht, BDDs für Zwischenergebnisse von Nachfolgerberechnungen und resultierende Zustandsmengen klein zu halten wurden Strategien zur Durchführung der Erreichbarkeitsanalyse entwickelt, die statt einer Breitensuche eine Kombination von Breiten- und Tiefensuche verwenden. In [166] wird die Dichte eines BDDs, als Anzahl der darin gespeicherten Minterme geteilt durch die Knotenanzahl des BDDs, definiert. Die Erreichbarkeitsanalyse wird mit einer Breitensuche gestartet. Wenn der BDD mit den in einer Iteration neu gefundenen Nachfolgerzuständen einen Grenzwert für die Größe überschreitet, wird in der nächsten Iteration nur eine Teilmenge dieser Zustände weiter exploriert. Die Teilmenge wird so ausgewählt, dass ihr BDD eine möglichst hohe Dichte aufweist. Dies wird mit der Erwartung positiver Effekte für die Größe des Resultats der Nachfolgerberechnung und des BDDs mit allen bereits gefundenen Zuständen durchgeführt. Durch spätere Nachfolgerberechnungen mit der Menge aller gefundenen Zustände wird wieder eine vollständige Exploration des Zustandsraums ermöglicht. Ein weiterer Algorithmus zur Erzeugung von BDDs für Teilmengen mit möglichst hoher Dichte wird in [165] vorgestellt.

Die Steuerung der Zustandsraumdurchmusterung (auch *guided search* genannt) durch vom Benutzer eingegebene Hinweise wird in [25, 167] behandelt. Hinweise sind Aussagen über erlaubte Werte von Eingabe- und Zustandsvariablen. Durch Kombination mit der Transitionsrelation beeinflussen sie die Ausführbarkeit von Transitionen, so dass nur noch eine Teilmenge der Transitionen ausführbar sein kann. Dies kann zu Vereinfachungen der

Transitionsrelation und kleineren Zwischenresultaten bei der Nachfolgerberechnung führen. In [41] werden Aktivitätsprofile für BDD-Knoten der Transitionsrelation aufgezeichnet. Diese werden zur Optimierung der Erreichbarkeitsanalyse durch Bildung von Teilmengen der Transitionsrelation, mit denen partielle Traversierungen des Zustandsraums durchgeführt werden, verwendet. Ein Verfahren zur Erreichbarkeitsanalyse, das ebenfalls partielle Traversierungen durchführt, welches aber mit dynamisch ermittelten Teilen der Transitionsrelation und Teilmengen der weiter zu explorierenden Zustände arbeitet, wird in [109] vorgeschlagen. Zur Bestimmung der Teilmengen wird die *Hamming-Distanz* zwischen in BDDs für Zustände gespeicherten Bitvektoren benutzt, mit dem Ziel BDDs für Zwischenergebnisse möglichst klein halten zu können.

Eine Modifikation zur Effizienzsteigerung der Breitensuche für Systeme aus lose gekoppelten Komponenten bei Verwendung disjunktiv partitionierter Transitionsrelationen wird in [39] präsentiert. Die Autoren schlagen vor, die erreichbaren Zustände komponentenweise durch separate Berechnungen von lokalen Fixpunkten für Komponenten bis zum Erreichen des globalen Fixpunktes zu ermitteln. In [133] wird ein Algorithmus für die symbolische Modellprüfung nebenläufiger Software bei Verwendung von BDDs präsentiert, der die Reduktion des Zustandsraums durch Halbordnung (engl. partial order reduction) [8, 56, 112, 157] ermöglicht. Bei ihm erfolgt die Exploration des Zustandsraums abwechselnd in zwei Phasen. In der ersten Phase werden komponentenweise Fixpunkte mit für die Ausnutzung der Reduktion des Zustandsraums durch Halbordnung angepassten Komponententransitionsrelationen berechnet. In Phase zwei erfolgt eine Berechnung von Nachfolgerzuständen mit einer Transitionsrelation, die Transitionen aller Komponenten beinhaltet. Techniken zur Verwendung der Reduktion durch Halbordnung nutzen die Kommutativität nebenläufig ausführbarer Transitionen aus, in dem sie versuchen aus Äquivalenzklassen von äquivalenten Transitionssequenzen nur repräsentative Transitionssequenzen zu explorieren. Der bei ihrer Verwendung für ein nebenläufiges System zu untersuchende Zustandsraum kann deutlich kleiner sein als der unreduzierte Zustandsraum. Weitere Arbeiten zur Reduktion durch Halbordnung bei der symbolischen Modellprüfung mit BDDs sind zum Beispiel [5, 146]. Andere Verfahren, die darauf abzielen die Verifikation von Systemen, für die eine exakte Erreichbarkeitsanalyse intraktabel ist, zu ermöglichen, berechnen Überapproximationen bzw. Unterapproximationen der Menge der erreichbaren Systemzustände [25, 43, 100, 101].

Chaining [169, 179] ist ein Ansatz, bei dem die Breitensuche zur effizienteren Verifikation nebenläufiger Systeme unter Benutzung partitionierter Transitionsrelationen modifiziert wurde. Bei der einfachen Verwendung partitionierter Transitionsrelationen für den in Abschnitt 2.1.6.3 präsentierten Algorithmus zur Vorwärtserreichbarkeitsanalyse, werden in einer Iteration von Nachfolgerberechnungen für alle Partitionen Nachfolgerberechnungen mit der gleichen zu explorierenden Zustandsmenge durchgeführt. Mit *Chaining* hingegen werden die bei der Nachfolgerberechnung mit einer Partition explorierten Zustände gleich zur Menge, der in dieser Iteration weiter zu explorierenden Zustände, hinzugefügt. Damit können diese Zustände bei Nachfolgerberechnungen mit der nächsten Partition der Transitionsrelation direkt mit berücksichtigt werden. Dies kann zu weniger Iterationen und deutlichen Laufzeitverbesserungen führen. Für die Effektivität von *Chaining* kann

die Reihenfolge, in der Partitionen für Nachfolgerberechnungen verwendet werden, wichtig sein. Verfahren zum Finden einer guten Reihenfolge der Anwendung der Partitionen zur Optimierung von *Chaining* werden in [44, 179] vorgestellt.

Vier verschiedene Verfahren zur Optimierung der Erreichbarkeitsanalyse werden in [96] vorgestellt. Diese versuchen durch Ausnutzen von positiven Effekten der Lokalität von Änderungen in Zuständen auf die Größe von BDDs, die Zwischenergebnisse klein zu halten. Bei zwei der Verfahren werden Werte von Variablen festgehalten und die Exploration des Zustandsraums wird nur mit Änderungen von Variablen durchgeführt, deren Werte nicht fixiert sind. Wenn ein Fixpunkt dieser Exploration erreicht wurde, wird einer größeren Menge an Variablen erlaubt ihre Werte zu ändern und die Exploration des Zustandsraums wird fortgesetzt. Dies wird wiederholt, bis alle Variablen ihre Werte ändern dürfen und alle erreichbaren Zustände exploriert wurden. Die beiden anderen Ansätze wurden für die Verwendung partitionierter Transitionsrelationen entworfen und geben verschiedene Reihenfolgen vor, in denen Partitionen der Transitionsrelation zur Berechnung der erreichbaren Zustände verwendet werden. Eine Methode kreist durch die Menge der Partitionen und führt für jede Partition solange Nachfolgerberechnungen durch, bis kein neuer Zustand mehr gefunden werden kann. Dann werden Nachfolgerberechnungen mit der nächsten Partition durchgeführt. Kann in einem Durchlauf aller Partitionen kein neuer Zustand gefunden werden, wurden alle erreichbaren Zustände gefunden. Der zweite Ansatz fährt, anstatt durch die Partitionen zu kreisen, bei Auffinden neuer Zustände wieder mit Nachfolgerberechnungen der ersten Partition fort.

Für die Benutzung von Varianten von MDDs [125] (siehe auch Abschnitt 2.5.4) und die Zustandsraumexploration asynchroner nebenläufiger Systeme mit Transitionen, die nur lokalen Einfluss auf den Zustandsvektor haben, wurde der *Saturation*-Algorithmus [46, 47, 48, 49, 151, 194] entwickelt. Er nutzt die durch die Lokalität von Transitionen resultierenden Freiheitsgrade in der Reihenfolge der Ausführung von Transitionen bei der Zustandsraumexploration aus. Dadurch weicht er von der sich durch Nachfolgerberechnungen bei vollständiger Verknüpfung mit allen Transitionen ergebenden Breitensuche ab. Bei ihm werden MDD-Knoten durch Berechnen von Fixpunkten für Mengen von Transitionen, die nur lokalen Einfluss haben, von unten nach oben im MDD saturiert. Ergeben sich dabei darunter liegende Knoten, die nicht mehr saturiert sind, werden zuerst diese saturiert, bevor Berechnungen für weiter oben im MDD liegende Knoten erfolgen. Dies führt dazu, dass möglichst wenige unsaturierte Knoten vorhanden sind, was zum Auffinden von vielen neuen Zuständen bei möglichst kleinen MDDs führen soll. Dadurch soll dem Problem, dass Zwischenergebnisse bei Berechnungen mit Entscheidungsdiagrammen häufig deutlich größer als Entscheidungsdiagramme für Resultate der Berechnungen sind, entgegengewirkt werden. Für diesen Ansatz präsentierte experimentelle Ergebnisse zeigen, dass daraus oft eine deutlich kleinere maximal benötigte Knotenanzahl und Laufzeitverbesserungen resultieren.

2.6 Beispielsystem

In diesem Abschnitt wird ein asynchrones nebenläufiges System mit Transitionslokalität (siehe Abschnitt 2.3) $M^2 = (S, S_0, R)$ vorgestellt, das im weiteren Verlauf dieser Arbeit für Beispiele zu Datenstrukturen und Algorithmen verwendet wird. Dieses System besteht aus zwei Komponenten, zu denen Zustandsübergangsdiagramme in Abbildung 2.12 angegeben sind. Beide Komponenten besitzen zwei lokale Zustände (s_0 und s_1) und der in der Abbildung hervorgehobene Zustand s_0 ist jeweils der Startzustand der Komponenten. Zur Repräsentation von Mengen von Zuständen und von Mengen von Transitionen dieses Systems für die BDD-basierte symbolische Modellprüfung, wird in der vorliegenden Arbeit eine binäre Kodierung von Zuständen verwendet. Dazu wird der aktuelle lokale Zustand von Komponente 1 (Komponente 2) durch die Belegung der Booleschen Variable x_1 (x_2) angegeben. Besitzt die Variable x_1 (x_2) in einem Systemzustand den Booleschen Wert 0, so befindet sich Komponente 1 (Komponente 2) in diesem Systemzustand in ihrem lokalen Zustand s_0. Ist der Wert der Variable x_1 (x_2) hingegen in einem Systemzustand der Boolesche Wert 1, dann befindet sich Komponente 1 (Komponente 2) in ihrem lokalen Zustand s_1. Der aktuelle globale Zustand des Systems wird durch Belegung einer Booleschen Variablen x_g angegeben. Der Wert dieser globalen Variablen wird durch eine Komponente nur beim Wechsel von ihrem Zustand s_1 in ihren Zustand s_0 verändert. Dabei setzt Komponente 1 den Wert der Variable x_g bei dieser Transition auf den Booleschen Wert 0, wohingegen Komponente 2 den Wert dieser Variable bei Ausführung ihrer entsprechenden Transition auf den Wert 1 setzt. Die zu diesen zur Repräsentation von aktuellen Zuständen verwendeten Booleschen Variablen korrespondierenden Booleschen Variablen für Nachfolgerzustände, sind die Variablen x'_1 (zu x_1), x'_2 (zu x_2) und x'_g (zu x_g).

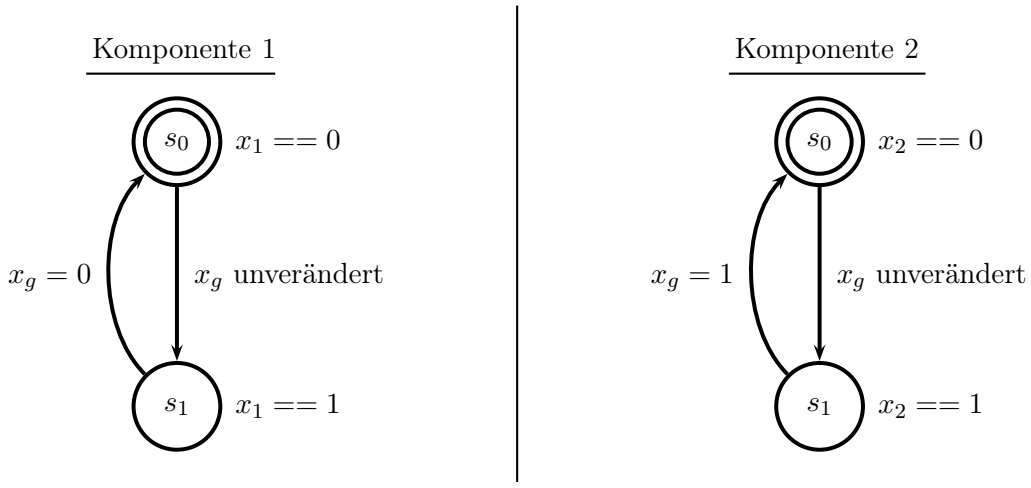

Abbildung 2.12: Zustandsübergangsdiagramme zum Beispielsystem aus zwei Komponenten.

Sei nun der Boolesche Wert 0 die Belegung der Variablen x_g zur Kodierung des globalen Zustands des Systems im einzigen Startzustand des Systems. Dann ergeben sich für das hier vorgestellte asynchrone nebenläufige System mit Transitionslokalität $M^2 = (S, S_0, R)$ bei Darstellung von Zuständen $s \in S$ durch Tupel $s = (\vec{g}, l_1, l_2)$, wie in Abschnitt 2.3 eingeführt, folgende Komponenten S, S_0 und R (die lokalen Zustände der Komponenten sind nachfolgend nicht durch ihre binäre Kodierung, sondern direkt durch den entsprechenden Zustandsbezeichner s_0 oder s_1 angegeben):

- $S = \{(0, s_0, s_0), (0, s_1, s_0), (0, s_0, s_1), (0, s_1, s_1), (1, s_0, s_0), (1, s_1, s_0), (1, s_0, s_1), (1, s_1, s_1)\}$
- $S_0 = \{(0, s_0, s_0)\}$
- $R = \{((0, s_0, s_0), (0, s_1, s_0)), ((0, s_0, s_0), (0, s_0, s_1)), ((0, s_1, s_0), (0, s_1, s_1)), ((0, s_1, s_0), (0, s_0, s_0)), ((0, s_0, s_1), (0, s_1, s_1)), ((0, s_0, s_1), (1, s_0, s_0)), ((0, s_1, s_1), (0, s_0, s_1)), ((0, s_1, s_1), (1, s_1, s_0)), ((1, s_0, s_0), (1, s_1, s_0)), ((1, s_0, s_0), (1, s_0, s_1)), ((1, s_1, s_0), (0, s_0, s_0)), ((1, s_1, s_0), (1, s_1, s_1)), ((1, s_0, s_1), (1, s_1, s_1)), ((1, s_0, s_1), (1, s_0, s_0)), ((1, s_1, s_1), (0, s_0, s_1)), ((1, s_1, s_1), (1, s_1, s_0))\}$

In der vorliegenden Arbeit werden Verbesserungen zur BDD-basierten symbolischen Modellprüfung präsentiert. Bei binärer Kodierung von Zuständen und Transitionen können Mengen dieser durch Boolesche charakteristische Funktionen (siehe Abschnitt 2.1.5) repräsentiert werden. Solche charakteristischen Funktionen für Mengen von Zuständen und Mengen von Transitionen werden bei der BDD-basierten symbolischen Modellprüfung durch BDDs repräsentiert.

Zum in diesem Abschnitt vorgestellten Beispiel sind BDDs zur Repräsentation aller Transitionen des Systems in Abbildung 2.13 dargestellt. In der Abbildung ist eine komponentenweise Partitionierung der gesamten Transitionsrelation, die aus einem BDD zur Repräsentation aller Transitionen von Komponente 1 und einem BDD zur Repräsentation aller Transitionen von Komponente 2 besteht, angegeben. Die Änderungen der lokalen Zustände der Komponenten werden in den BDDs für Komponente 1 durch Knoten für die Variablen x_1 und x_1' und für Komponente 2 durch Knoten für die Variablen x_2 und x_2' kodiert. Die entsprechenden Bereiche der BDDs sind in Abbildung 2.13 als *Ausführbereich* eingezeichnet. Für die Änderungen der lokalen Zustände der anderen Komponente, für die ein BDD keine Transitionen enthält, sind in den BDDs Identitätsmuster (siehe Abschnitt 2.5.3) angegeben. Diese wurden ebenfalls in Abbildung 2.13 gekennzeichnet. Ein BDD zur Repräsentation des Anfangszustands $s_0 = (0, s_0, s_0)$ des Systems ist in Abbildung 2.14 angegeben. BDDs zur Repräsentation von Zustandsmengen besitzen wie dieser BDD nur Knoten für Variablen für aktuelle Zustände. In den Abbildungen vorhandene Markierungen die hier nicht beschrieben wurden, werden in den in den nachfolgenden Abschnitten angegebenen Beispielen verwendet.

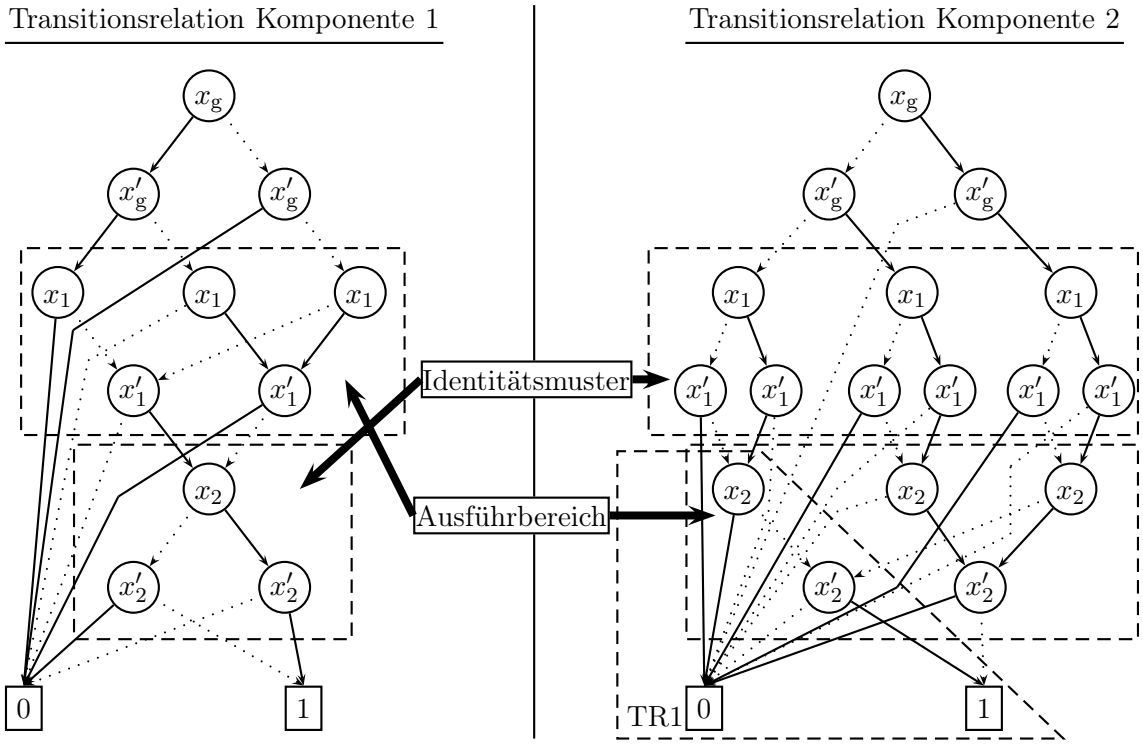

Abbildung 2.13: BDDs zur Repräsentation aller Transitionen von Komponente 1 bzw. von Komponente 2, jeweils mit der Variablenordnung $<_{level_3}$ mit $x_g <_{level_3} x'_g <_{level_3} x_1 <_{level_3} x'_1 <_{level_3} x_2 <_{level_3} x'_2$.

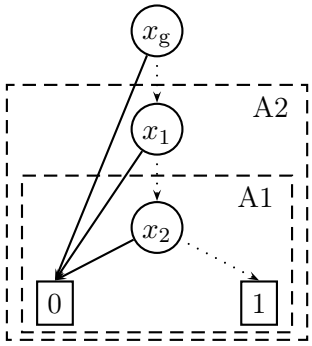

Abbildung 2.14: BDD zur Repräsentation des Anfangszustands $s_0 = (0, s_0, s_0)$, mit der Variablenordnung $<_{level_3}$ mit $x_g <_{level_3} x'_g <_{level_3} x_1 <_{level_3} x'_1 <_{level_3} x_2 <_{level_3} x'_2$.

3 Kombinierte Berechnung des relationalen Produkts

Eine zentrale Operation bei der BDD-basierten symbolischen Modellprüfung ist die wiederholt durchzuführende Berechnung von Nachfolgerzuständen bzw. Vorgängerzuständen einer Zustandsmenge. Die Berechnung der Menge der unmittelbaren Nachfolgerzustände einer Zustandsmenge wurde für die BDD-basierte Modellprüfung in Abschnitt 2.5.1 durch das relationale Produkt $\exists\vec{x}[R(\vec{x},\vec{x}') \wedge S(\vec{x})]\{\vec{x}' \leftarrow \vec{x}\}$ beschrieben. Im relationalen Produkt sind $R(\vec{x},\vec{x}')$ und $S(\vec{x})$ charakteristische Funktionen der binär kodierten Transitionsrelation ($R(\vec{x},\vec{x}')$), bzw. der binär kodierten Menge der zu explorierenden Zustände ($S(\vec{x})$). Diese charakteristischen Funktionen werden bei der BDD-basierten Modellprüfung durch BDDs repräsentiert. Beim ursprünglich verwendeten Ansatz zur Berechnung das relationalen Produkts wurden die drei im relationalen Produkt enthaltenen Operationen durch separate aufeinander folgende BDD-Traversierungen durchgeführt. Dabei erfolgte zuerst die UND-Verknüpfung der Transitionsrelation mit der zu explorierenden Zustandsmenge ($R(\vec{x},\vec{x}') \wedge S(\vec{x})$). Anschließend wurde die existentielle Quantifizierung bezüglich der Variablen zur Kodierung der aktuellen Zustände ($\exists\vec{x}$) für das Resultat der UND-Verknüpfung berechnet. Für das daraus resultierende Ergebnis wurde danach die Umbenennung der Knoten für Variablen für Nachfolgerzustände in Knoten für deren korrespondierende Variablen für aktuelle Zustände ($\{\vec{x}' \leftarrow \vec{x}\}$) (siehe auch Abschnitt 2.5.1) ausgeführt. Durch die sequentielle Berechnung des relationalen Produkts kann sich eine hohe Gesamtlaufzeit, die sich aus der Summe der Laufzeiten der einzelnen Operationen zusammen setzt, ergeben. Außerdem können BDDs für während der sequentiellen Berechnung auftretende Zwischenergebnisse deutlich größer sein, als das Endergebnis der Berechnung des relationalen Produkts selbst.

Die Berechnung des relationalen Produkts kann aber auch durch gemeinsame kombinierte Berechnung der im relationalen Produkt enthaltenen Operationen erfolgen. In [142] wurde ein Algorithmus vorgestellt, der die UND-Verknüpfung und die existentielle Quantifizierung gemeinsam berechnet. Für verschachtelte Variablenordnungen, in denen korrespondierende Boolesche Variablen für aktuelle Zustände und Nachfolgerzustände in der Variablenordnung immer direkt benachbart sind, wurde außerdem in [8] ein Algorithmus präsentiert, der das relationale Produkt in einer BDD-Traversierung berechnet.

Im nachfolgenden Abschnitt wird ebenfalls ein Algorithmus zur kombinierten Berechnung des relationalen Produkts bei verschachtelten Variablenordnungen vorgestellt. Eine Abgrenzung dieses Algorithmus zu den beiden zuvor beschriebenen Verfahren ist in Abschnitt 3.5 zu finden. Der Vorteil verschachtelter Variablenordnungen ist, dass für diese die im relationalen Produkt enthaltene Umbenennungsoperation in linearer Zeit durchgeführt werden kann. Der Grund ist, dass Variablen zur Kodierung von Nachfolgerzuständen

hier durch direktes Umbenennen in die entsprechenden benachbarten Variablen für aktuelle Zustände umbenannt werden können (siehe Abschnitt 2.5.3). Wie die Ergebnisse der im Rahmen der vorliegenden Arbeit durchgeführten Verifikationsexperimente zeigen (siehe Abschnitt 8.5.1), führt der im nachfolgenden Abschnitt vorgestellte Algorithmus zur kombinierten Berechnung des relationalen Produkts im Vergleich mit der sequentiellen Berechnung des relationalen Produkts zu sehr großen Laufzeitverbesserungen und oft auch zu deutlichen Verringerungen des maximal benötigten Speicherbedarfs.

3.1 Algorithmus zur kombinierten Berechnung des relationalen Produkts

In diesem Abschnitt wird für verschachtelte Variablenordnungen ein Algorithmus zur Berechnung des relationalen Produkts für die Nachfolgerberechnung $\exists\vec{x}[R(\vec{x},\vec{x}')\wedge S(\vec{x})]\{\vec{x}' \leftarrow \vec{x}\}$ vorgestellt, der die im relationalen Produkt enthaltenen Operationen auf einmal in einer BDD-Traversierung berechnet. Pseudocode dieses Algorithmus ist in Algorithmus 9 angegeben. Das relationale Produkt wird vom Algorithmus, nach dem bei Operationen mit BDDs üblicherweise verwendeten und in Abschnitt 2.2.4 beschriebenen rekursiven Berechnungsschema, berechnet. Bei diesem erfolgt in den Rekursionsschritten eines Algorithmus eine Aufspaltung der durchzuführenden Berechnung in zwei Teilprobleme aus einfacheren Kofaktoren, die aus der Belegung einer Booleschen Variablen mit den beiden Booleschen Werten resultieren. Für die Teilprobleme ermittelte Ergebnisse werden anschließend zum Ergebnis eines Rekursionsschritts kombiniert. Vor rekursiven Aufrufen des Algorithmus für Teilprobleme findet in Rekursionsschritten des Algorithmus wie beim APPLY-Algorithmus (siehe Algorithmus 2 in Abschnitt 2.2.4) eine Prüfung auf Terminalfälle statt. Zusätzlich wird bei nicht vorhandenem Terminalfall vor weiteren rekursiven Aufrufen geprüft, ob das Ergebnis der aktuell zu berechnenden Operation bereits vorher ermittelt wurde und noch in der Ergebnistabelle (engl. Computed Table, siehe auch Abschnitt 2.2.3) gespeichert ist. In einem Rekursionsschritt des Algorithmus erfolgt die Aufspaltung in Teilprobleme bezüglich der Variable aus den Variablen der Wurzelknoten der BDDs R und S, die weiter oben in der aktuell verwendeten Variablenordnung angeordnet ist. Diese Variable wird auch als *Top-Variable* eines Rekursionsschritts bezeichnet. Die Korrektheit der im Algorithmus verwendeten Aufspaltung in Teilprobleme und der Kombination von ermittelten Teilergebnissen zum Ergebnis eines Rekursionsschritts des Algorithmus wird in Abschnitt 3.2 gezeigt. Im Algorithmus wird bei Rekursionsschritten mit einer Aufspaltung bezüglich einer Top-Variablen für aktuelle Zustände und bei Rekursionsschritten mit einer Aufspaltung bezüglich einer Top-Variablen für Nachfolgerzustände unterschiedlich verfahren. Der Algorithmus geht dabei bei der Aufspaltung des relationalen Produkts $\exists\vec{x}[R(\vec{x},\vec{x}') \wedge S(\vec{x})]\{\vec{x}' \leftarrow \vec{x}\}$ in Teilprobleme, die jeweils aus der Belegung einer Variablen v mit einem der beiden Booleschen Werte resultieren, wie in den nachfolgend aufgeführten und im Beweis zu Satz 3.2 in Abschnitt 3.2 bewiesenen Gleichungen vor:

- υ Variable für aktuelle Zustände ($\upsilon \in X$):

$$\begin{aligned}\exists\vec{x}[R(\vec{x},\vec{x}') \wedge S(\vec{x})]\{\vec{x}' \leftarrow \vec{x}\} =&(\exists\vec{x}(R(\vec{x},\vec{x}')|_{\upsilon=1} \wedge S(\vec{x})|_{\upsilon=1}))\{\vec{x}' \leftarrow \vec{x}\} \vee \\ &(\exists\vec{x}(R(\vec{x},\vec{x}')|_{\upsilon=0} \wedge S(\vec{x})|_{\upsilon=0}))\{\vec{x}' \leftarrow \vec{x}\}\end{aligned}$$

- υ Variable für Nachfolgerzustände ($\upsilon \in X'$):

$$\begin{aligned}\exists\vec{x}[R(\vec{x},\vec{x}') \wedge S(\vec{x})]\{\vec{x}' \leftarrow \vec{x}\} =&(\upsilon)\{\vec{x}' \leftarrow \vec{x}\} \cdot (\exists\vec{x}(R(\vec{x},\vec{x}')|_{\upsilon=1} \wedge S(\vec{x})|_{\upsilon=1})) \\ &\{\vec{x}' \leftarrow \vec{x}\} \vee (\overline{\upsilon})\{\vec{x}' \leftarrow \vec{x}\} \cdot (\exists\vec{x}(R(\vec{x},\vec{x}')|_{\upsilon=0} \\ &\wedge S(\vec{x})|_{\upsilon=0}))\{\vec{x}' \leftarrow \vec{x}\}\end{aligned}$$

Algorithmus 9: Kombinierte Berechnung des relationalen Produkts für die Nachfolgerberechnung $\exists\vec{x}[R(\vec{x},\vec{x}') \wedge S(\vec{x})]\{\vec{x}' \leftarrow \vec{x}\}$ mit BDDs für verschachtelte Variablenordnungen.

```
1  RelProduct(ROBDD R, ROBDD S)
2  if IsTerminalCase(R, S) then
3  |  return TerminalCase(R, S);
4  else
5  |  if ComputedTableHasEntry(RelProd, R, S) then
6  |  |  return ComputedTable(RelProd, R, S);
7  |  else
8  |  |  υ = TopVariable(VariableRoot(R), VariableRoot(S));
9  |  |  R1 = RelProduct(R|υ=1, S|υ=1);
10 |  |  R0 = RelProduct(R|υ=0, S|υ=0);
11 |  |  if R0 == R1 then
12 |  |  |  return R1;
13 |  |  else
14 |  |  |  if IsCurrentStateVar(υ) then
15 |  |  |  |  Result = APPLY(Or, R0, R1);
16 |  |  |  else
17 |  |  |  |  Result = FindOrAddUniqueTable(GetCurrentStateVar(υ), R0, R1);
18 |  |  InsertComputedTable(RelProd, R, S, Result);
19 |  |  return Result;
```

Im weiteren Verlauf dieses Abschnitts wird der Algorithmus detaillierter beschrieben. Als Übergabeparameter besitzt er einen BDD R zur Repräsentation eines Kofaktors des BDDs für die Transitionsrelation und einen BDD S zur Repräsentation eines Kofaktors des BDDs mit der zu explorierenden Zustandsmenge. In einem Aufruf des Algorithmus wird zuerst geprüft, ob ein Terminalfall der rekursiven Berechnung vorliegt (Zeile 2). Wurde

ein Terminalfall erreicht, kann das Ergebnis der im aktuellen Rekursionsschritt durchzuführenden Berechnung direkt ermittelt und zurückgegeben werden (Zeile 3). Algorithmus 9 verwendet andere Terminalfälle als der in [8] zur kombinierten Berechnung des relationalen Produkts vorgestellte Algorithmus. In [8] wurden als Terminalfälle nur die Fälle verwendet, bei denen in einem Rekursionsschritt mindestens einer der Wurzelknoten der BDDs R oder S ein Terminalknoten für den Booleschen Wert 0 ist, oder in dem beide Wurzelknoten Terminalknoten für den Booleschen Wert 1 sind. Aus der gemeinsamen Ausführung der Operationen des relationalen Produkts und der Ausnutzung von daraus resultierenden Eigenschaften lassen sich aber neue Terminalfälle ableiten. Diese können zu einer früheren Terminierung der rekursiven Berechnung führen. Zur Berechnung der Rückgabewerte von Terminalfällen ruft Algorithmus 9 die Funktion *TerminalCase()* auf, von der Pseudocode in Algorithmus 10 zu finden ist. In dieser Funktion sind die im in diesem Abschnitt vorgestellten Algorithmus verwendeten Terminalfälle und deren Rückgabewerte zu finden.

Algorithmus 10: Funktion *TerminalCase()*, die bei einem in Algorithmus 9 aufgetretenen Terminalfall den BDD mit dem Ergebnis des Terminalfalls berechnet und zurückgibt.

```
1 TerminalCase(ROBDD R, ROBDD S)
2 if IsTerminalVertexZero(R) || IsTerminalVertexZero(S) then
3 |   return BDDTerminalVertexZero;
4 if IsTerminalVertexOne(R) || R==S then
5 |   return BDDTerminalVertexOne;
```

Ein Terminalfall liegt hier ebenfalls vor, wenn der Wurzelknoten eines der beiden BDDs R oder S ein Terminalknoten für den Booleschen Wert 0 ist. Wird bei der Berechnung des relationalen Produkts und bei der dabei durchgeführten Traversierung der beteiligten BDDs in einem der BDDs ein Terminalknoten erreicht, dann entspricht der Boolesche Wert des Terminalknotens der Auswertung der im BDD gespeicherten Funktion für alle Belegungen der Booleschen Eingabevariablen, die dem beim Traversieren genommenen Pfad von der Wurzel bis zu diesem Terminalknoten entsprechen. Im relationalen Produkt ist außer der existentiellen Quantifizierung und der Umbenennung von Variablen die UND-Verknüpfung der BDDs R und S durchzuführen. Wird in einem der beteiligten BDDs beim rekursiven Traversieren ein Terminalknoten für den Booleschen Wert 0 erreicht, ist das Ergebnis der UND-Verknüpfung für alle Belegungen der bei der aktuellen rekursiven Berechnung noch nicht betrachteten Eingabevariablen der Boolesche Wert 0. Dies gilt unabhängig von Wert eines im anderen BDD möglicherweise erst später erreichten Terminalknotens. Ist das Ergebnis der UND-Verknüpfung in einem Rekursionsschritt für alle Belegungen der noch zu evaluierenden Eingabevariablen der Boolesche Wert 0, ist der Wert 0 auch der Rückgabewert des relationalen Produkts für diesen Rekursionsschritt nach Anwendung der noch durchzuführenden existentiellen Quantifizierungen und Umbenennungen. Damit ist das Ergebnis des relationalen Produkts für einen Rekursionsschritt

der Boolesche Wert 0, wenn in einem der beiden BDDs ein Terminalknoten für den Booleschen Wert 0 erreicht wird. Wenn im Algorithmus aus [8] in einem der beiden BDDs ein Terminalknoten für den Wert 1 erreicht wird, wird die rekursive Berechnung fortgesetzt. Ein Terminalfall tritt in diesem Algorithmus hier erst auf, wenn auch im zweiten BDD ein Terminalknoten erreicht wird. Der Boolesche Wert des Resultats der UND-Verknüpfung kann durch das Erreichen eines Terminalknotens für den Wert 1 in einem BDD noch nicht entschieden werden, weshalb das Ergebnis der UND-Verknüpfung in einem solchen Fall vom Wert des im zweiten BDD erreichten Terminalknotens abhängig ist.

Im hier vorgestellten Algorithmus werden im Vergleich zum in [8] für das relationale Produkt präsentierten Algorithmus optimierte Terminalfälle verwendet, die eine frühere Terminierung der rekursiven Berechnung ermöglichen können. Bei den neuen Terminalfällen wird die rekursive Berechnung bereits beendet, falls im BDD R für die Transitionsrelation ein Terminalknoten für den Wert 1 erreicht wird oder wenn die BDDs R und S gleich sind (siehe Zeile 4 in Algorithmus 10). Da in der Funktion *TerminalCase()* vor dem Überprüfen auf das Vorliegen dieser neuen Terminalfälle schon getestet wurde, ob der Wurzelknoten eines der beiden BDDs ein Terminalknoten für den Wert 0 ist, können die BDDs R und S bei Erreichen der neuen Terminalfälle keine Terminalknoten für den Wert 0 sein. Damit ergibt sich für die neuen Terminalfälle der Terminalknoten für den Wert 1 (siehe Zeile 5 in Algorithmus 10) als Rückgabewert. Auf die Korrektheit der neuen Terminalfälle wird in Abschnitt 3.3 eingegangen.

Liegt im aktuellen Rekursionsschritt kein Terminalfall vor, wird getestet, ob das Ergebnis der im aktuellen Rekursionsschritt durchzuführenden Berechnung bereits vorher ermittelt wurde und noch in der Ergebnistabelle gespeichert ist (Zeile 5 in Algorithmus 9). Beim entsprechenden Zugriff auf die Ergebnistabelle werden die Art der Operation (hier relationales Produkt, *RelProd*) und die Wurzelknoten der beiden beteiligten BDDs (R, S) als Übergabeparameter verwendet. Diese werden in den Einträgen der Ergebnistabelle mit abgespeichert um sicherzustellen, dass bei Anfragen an die Ergebnistabelle nur korrekte Ergebnisse für die gleiche Operationsart und die gleichen Operanden zurückgeliefert werden. Wird das zu berechnende Resultat in der Ergebnistabelle gefunden, wird es als Ergebnis des Rekursionsschritts zurückgegeben (Zeile 6). Liegt in einem Rekursionsschritt kein Terminalfall vor und konnte das Ergebnis der zu berechnenden Operation auch nicht in der Ergebnistabelle gefunden werden, wird die Berechnung des relationalen Produkts durch Aufspaltung in Teilprobleme und weitere rekursive Aufrufe des Algorithmus fortgesetzt. Dazu wird aus den Variablen der Wurzelknoten der beiden BDDs R und S die Top-Variable eines Rekursionsschritts ermittelt, die die Variable aus diesen beiden Variablen ist, die weiter oben in der aktuell verwendeten Variablenordnung angeordnet ist (Zeile 8). Bezüglich dieser wird anschließend die weitere Aufspaltung in Teilprobleme durchgeführt. Zur Aufspaltung in Teilprobleme werden wie in den oben angegebenen Gleichungen zu sehen, Kofaktoren der beiden BDDs R und S bezüglich Belegungen der Top-Variable mit den Booleschen Werten 0 bzw. 1 bestimmt. Mit diesen wird Algorithmus 9 anschließend rekursiv zur Ermittlung der Lösungen der Teilprobleme erneut aufgerufen (siehe Zeilen 9 und 10).

Wurden die Ergebnisse für die einfacheren Teilprobleme ermittelt, wird das Ergebnis des relationalen Produkts für den aktuellen Rekursionsschritt aus den entsprechenden Teilergebnissen R_0 bzw. R_1 berechnet. Dabei wird auch die im relationalen Produkt bezüglich der Variablen für aktuelle Zustände durchzuführende existentielle Quantifizierung und die Umbenennung von Variablen für Nachfolgerzustände, in deren korrespondierenden Variablen für aktuelle Zustände, durchgeführt. Dies erfolgt wie in den oben für die Aufspaltung in Teilprobleme aufgeführten Gleichungen angegeben. In Algorithmus 9 werden drei Fälle zur Kombination der beiden rekursiv ermittelten Teilresultate R_0 und R_1 zum Resultat eines Rekursionsschritts unterschieden. Sind die BDDs der beiden Teilergebnisse gleich ($R_0 == R_1$, siehe Zeile 11), führt die Belegung der Top-Variable des aktuellen Rekursionsschritts für beide Booleschen Werte zum gleichen Teilergebnis. Daher ist hier im BDD des Resultats aufgrund der Reduktionsregeln keine Verzweigung zum passenden Ergebnis in Abhängigkeit des Booleschen Wertes der Top-Variable nötig. Ebenfalls führen die existentielle Quantifizierung oder eine durchzuführende Umbenennung für einen solchen Rekursionsschritt nicht zur Erzeugung eines neuen BDD-Knotens. Deshalb kann einer der Ergebnis-BDDs der rekursiven Aufrufe (z.B. R_1, siehe Zeile 12) als Resultat zurückgegeben werden. Wenn sich die durch die rekursiven Aufrufe ermittelten Ergebnis-BDDs R_0 und R_1 unterscheiden, sind bei der Berechnung des Rückgabewerts zwei verschiedene Fälle zu berücksichtigen. Handelt es sich bei der Top-Variablen v eines Rekursionsschritts um eine Variable zur Kodierung eines aktuellen Zustands ($v \in X$), wird die im relationalen Produkt bezüglich der Variablen für aktuelle Zustände auszuführende existentielle Quantifizierung direkt durchgeführt. Dazu werden die BDDs der Teilresultate R_0 und R_1 durch Veroderung zum Ergebnis des aktuellen Rekursionsschritts kombiniert (Zeile 15). Dies ist korrekt, da die existentielle Quantifizierung einer Funktion bezüglich einer Variablen durch die Veroderung der Kofaktoren, die sich bei Belegung dieser Variablen mit dem Wert 0 und dem Wert 1 ergeben, definiert ist. Zur Durchführung der Veroderung kann der in Abschnitt 2.2.4 vorgestellte *APPLY*-Algorithmus verwendet werden.

Ist die Top-Variable v des aktuellen Rekursionsschritts eine Variable für Nachfolgerzustände ($v \in X'$), wird in Algorithmus 9 direkt die im relationalen Produkt ebenfalls zu berechnende Umbenennung von Knoten für Variablen für Nachfolgerzustände in Knoten für deren korrespondierende Variablen für aktuelle Zustände durchgeführt ($(v)\{\vec{x}' \leftarrow \vec{x}\}$ und $\overline{v}\{\vec{x}' \leftarrow \vec{x}\}$ in der für eine Top-Variable für Nachfolgerzustände oben angegebenen Gleichung, siehe Zeile 17). Dadurch werden vom Algorithmus in diesem Fall keine Knoten für Variablen zur Kodierung von Nachfolgerzuständen, sondern immer direkt Knoten für deren korrespondierende Variablen für aktuelle Zustände, erzeugt. Das Ergebnis eines Rekursionsschritts mit einer Top-Variablen für Nachfolgerzustände ist ein BDD, der als Wurzelknoten einen Knoten für die zur Variable für Nachfolgerzustände v korrespondierende Variable für aktuelle Zustände (*GetCurrentStateVar(v)*) besitzt und dessen Nachfolgerknoten die BDDs der Teilresultate R_0 bzw. R_1 sind. Der Resultatsknoten wird durch Aufruf der Funktion *FindOrAddUniqueTable()*, die hier wie in Algorithmus 3 in Abschnitt 2.2.4 angegeben implementiert ist, erzeugt (siehe Zeile 17). Die für Top-Variablen für Nachfolgerzustände notwendige Umbenennung in die korrespondierende Variable für aktuelle Zustände ist für verschachtelte Variablenordnungen auf diese Weise einfach möglich.

In solchen Variablenordnungen sind Knoten für die korrespondierenden Variablen immer direkt benachbart. Nach der Berechnung des Resultats wird dieses zur Effizienzsteigerung in der Ergebnistabelle abgespeichert (siehe Zeile 18) und anschließend zurückgegeben. Aufgrund des zusätzlichen Aufrufs des APPLY-Algorithmus bei der Ermittlung des Resultats eines Rekursionsschritts (Zeile 15), besitzt der Algorithmus im schlimmsten Fall exponentielle Laufzeit in der Größe der Eingabe-ROBDDs [197].

Im Folgenden wird die Vorgehensweise von Algorithmus 9 mit Hilfe des in Abschnitt 2.6 vorgestellten Beispielsystems veranschaulicht. Dort sind in Abbildung 2.13 zwei BDDs zur Repräsentation der Transitionsrelation der beiden im Beispielsystem vorhandenen Komponenten dargestellt und in Abbildung 2.14 ist ein BDD zur Repräsentation des Anfangszustands des Systems zu sehen. Die Vorgehensweise des Algorithmus wird nachfolgend für eine Berechnung des relationalen Produkts mit dem in Abbildung 2.13 angegebenen BDD, der die Transitionen von Komponente 2 repräsentiert und dem BDD zur Repräsentation des Anfangszustands veranschaulicht. Da für die Berechnung des relationalen Produkts mit den dort dargestellten BDDs viele rekursive Aufrufe nötig wären, wird das Vorgehen des Algorithmus hier aus Gründen der Übersichtlichkeit für Ausschnitte dieser beiden BDDs beschrieben. Die Ausschnitte der BDDs, die im hier veranschaulichten Beispiel im ersten Aufruf des Algorithmus *RelProduct()* als Übergabeparameter verwendet werden, sind in Abbildung 3.1 oben dargestellt. Außerdem sind die Ausschnitte in den entsprechenden BDDs in Abschnitt 2.6 über das Verifikationsbeispiel durch Umrandungen, die im BDD mit der Transitionsrelation mit *TR1* und im BDD für den Anfangszustand mit *A1* beschriftet sind, gekennzeichnet. Im Folgenden wird die Vorgehensweise von Algorithmus 9 bei der Berechnung des Resultats des relationalen Produkts mit diesen Ausschnitten der BDDs als Eingabeparametern beschrieben. Dies geschieht mit Hilfe der Veranschaulichung in Abbildung 3.1. Das Vorgehen des Algorithmus wird dabei ohne die Benutzung der Ergebnistabelle und damit ohne Beschleunigung des Algorithmus durch Rückgabe bereits berechneter und noch in der Ergebnistabelle gespeicherter Teilresultate erklärt.

Beim Aufruf von Algorithmus *RelProduct()* mit den beiden in Abbildung 3.1 oben angegebenen BDDs, liegt keiner der in Algorithmus 10 aufgeführten Terminalfälle vor. Daher erfolgt die weitere Berechnung des relationalen Produkts durch Aufspaltung in Teilprobleme, wozu in Zeile 8 von Algorithmus 9 die Top-Variable des aktuellen Rekursionsschritts bestimmt wird. Bezüglich der als Top-Variable des ersten Aufrufs des Algorithmus ermittelten Variablen x_2 werden nachfolgend Kofaktoren der aktuellen BDDs R und S, für eine Belegung der Variablen x_2 mit den beiden Booleschen Werten, ermittelt. Die zu diesen Kofaktoren korrespondierenden BDDs $R|_{x_2=0}$, $R|_{x_2=1}$, $S|_{x_2=0}$ und $S|_{x_2=1}$ sind in Abbildung 3.1 im für den *1. Aufruf* des Algorithmus vorhandenen umrandeten Bereich angegeben. Mit diesen Kofaktoren als Übergabeparametern wird der Algorithmus rekursiv erneut aufgerufen. Dabei wird zuerst der rekursive Aufruf des Algorithmus mit den Kofaktoren für eine Belegung der Variablen x_2 mit dem Booleschen Wert 1 durchgeführt (siehe Zeile 9). Im dazu durchgeführten *2. Aufruf* des Algorithmus ist direkt ein Terminalfall mit dem Terminalknoten für den Wert 0 als Rückgabewert vorhanden, da die Kofaktoren $R|_{x_2=1}$ und $S|_{x_2=1}$ beide Terminalknoten für den Wert 0 sind. Anschließend erfolgt im *1. Aufruf* des Algorithmus ein rekursiver Aufruf des Algorithmus mit

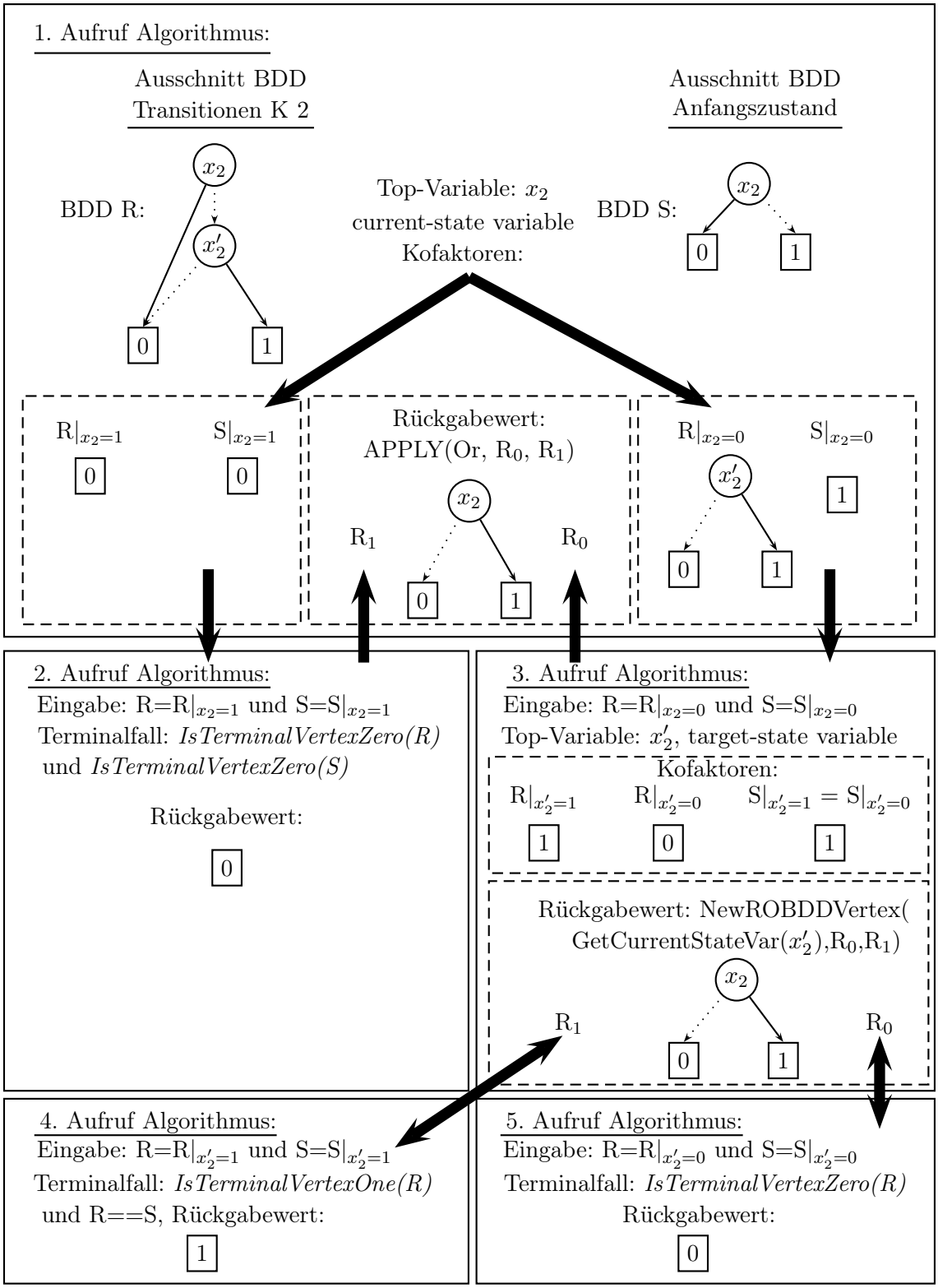

Abbildung 3.1: Beispiel zur Bildberechnung mit dem Algorithmus zur kombinierten Berechnung des relationalen Produkts.

den Kofaktoren für eine Belegung der Top-Variable x_2 mit dem Wert 0, zur Ermittlung des Ergebnisses des entsprechenden Teilproblems. Die Top-Variable des entsprechenden *3. Aufrufs* des Algorithmus, in dem nicht direkt ein Terminalfall vorliegt, ist die Variable x'_2. Bei der Berechnung der Kofaktoren bezüglich der Top-Variable x'_2, ergeben sich die in der in Abbildung 3.1 für den *3. Aufruf* des Algorithmus angegebenen Umrandung dargestellten Kofaktoren. Da in diesem Rekursionsschritt kein Terminalfall vorliegt, werden weitere rekursive Aufrufe des Algorithmus mit Kofaktoren bezüglich der Top-Variable x'_2 durchgeführt. In den dazu für eine Belegung der Variable mit beiden Booleschen Werten ausgeführten rekursiven Aufrufen liegen jeweils direkt Terminalfälle vor.

Im *4. Aufruf* des Algorithmus, der als Übergabeparameter die Kofaktoren bezüglich einer Belegung der Variable x'_2 mit dem Booleschen Wert 1 besitzt, ist der Wurzelknoten des BDDs R der Terminalknoten für den Wert 1 und es entspricht der BDD R dem BDD S. Daher sind hier beide in Zeile 4 in Algorithmus 10 angegebenen Terminalfälle erfüllt und der Terminalknoten für den Booleschen Wert 1 ist der Rückgabewert dieses Rekursionsschritts. Der *5. Aufruf* des Algorithmus bekommt den Terminalknoten für den Booleschen Wert 0 als Kofaktor für eine Belegung der Variable x'_2 mit dem Wert 0 ($R|_{x'_2=0}$) übergeben. Damit tritt hier ebenfalls direkt ein Terminalfall auf (siehe Zeile 2 in Algorithmus 10), der zum Terminalknoten für den Booleschen Wert 0 als Rückgabewert führt. Im *3. Aufruf* des Algorithmus werden die vom *4. Aufruf* und vom *5. Aufruf* erhaltenen Rückgabewerte anschließend zum Rückgabewert des Rekursionsschritts kombiniert. Bei der Top-Variablen x'_2 des *3. Aufrufs* handelt es sich um eine Variable zur Kodierung von Nachfolgerzuständen und die rekursiv ermittelten Teilergebnisse sind nicht gleich. Daher findet hier direkt die Umbenennung der Variable x'_2 für Nachfolgerzustände in die korrespondierende Variable x_2 für aktuelle Zustände statt und es wird ein neuer Knoten für die Variable x_2 mit den ermittelten Teilresultaten als Nachfolgerknoten erzeugt (*FindOrAddUniqueTable(GetCurrentStateVar(*v*),* R_0*,* R_1*)*). Abschließend wird der Rückgabewert des *1. Aufrufs* des Algorithmus berechnet. Da sich die rekursiv ermittelten Teilergebnisse für diesen Rekursionsschritt unterscheiden und es sich bei der Top-Variablen x_2 dieses Rekursionsschritts um eine Variable für aktuelle Zustände handelt, wird hier direkt die im relationalen Produkt notwendige existentielle Quantifizierung bezüglich der Variablen für aktuelle Zustände durchgeführt. Dazu werden die Teilresultate R_0 und R_1 verodert (*APPLY(Or,*R_1*,*R_0*)*). Der Ergebnis-BDD der Veroderung ist das Resultat des relationalen Produkts für das angegebene Teilproblem, das vom 1. Aufruf des Algorithmus zur kombinierten Berechnung des relationalen Produkts zurückgegeben wird.

3.2 Korrektheit des Algorithmus

Der in Abschnitt 3.1 vorgestellte Algorithmus berechnet das gesamte relationale Produkt für verschachtelte Variablenordnungen in einer BDD-Traversierung. Im Folgenden wird die Korrektheit des Vorgehens des Algorithmus bei der rekursiven Aufspaltung der in einem Rekursionsschritt durchzuführenden Berechnung in Teilprobleme und der anschließenden Kombination der ermittelten Teilergebnisse zum Rückgabewert eines Rekursionsschritts gezeigt.

Dazu wird zunächst eine Berechnungsvorschrift zur Durchführung der existentiellen Quantifizierung, bei der im Algorithmus verwendeten Aufspaltung in Teilprobleme aus einfacheren Kofaktoren, bewiesen. Im Beweis wird ausgenutzt, dass für die existentielle Quantifizierung $\exists_{x_i} f$ einer booleschen Funktion f bezüglich einer Variablen x_i (siehe Definition 2.38) die Distributivität hinsichtlich der Disjunktion ($\vee$) gilt [144].

Lemma 3.1. *Gegeben sei die Teilberechnung $\exists\vec{x}[R(\vec{x},\vec{x}')\wedge S(\vec{x})]$ des relationalen Produkts $\exists\vec{x}[R(\vec{x},\vec{x}')\wedge S(\vec{x})]\{\vec{x}'\leftarrow\vec{x}\}$, wie es in Definition 2.35 eingeführt wurde. Sei außerdem υ eine Boolesche Variable aus einer der dort definierten Variablenmengen X und X'. Dann kann die Berechnung von $\exists\vec{x}[R(\vec{x},\vec{x}')\wedge S(\vec{x})]$ in Abhängigkeit davon, ob υ eine Variable aus X oder X' ist, folgendermaßen aufgespalten werden:*

$$\exists\vec{x}[R(\vec{x},\vec{x}')\wedge S(\vec{x})] = \begin{cases} \exists\vec{x}(R(\vec{x},\vec{x}')|_{\upsilon=1}\wedge S(\vec{x})|_{\upsilon=1})\vee & \\ \exists\vec{x}(R(\vec{x},\vec{x}')|_{\upsilon=0}\wedge S(\vec{x})|_{\upsilon=0}) & \upsilon\in X \\ & \\ \upsilon\cdot(\exists\vec{x}(R(\vec{x},\vec{x}')|_{\upsilon=1}\wedge S(\vec{x})|_{\upsilon=1}))\vee & \\ \overline{\upsilon}\cdot(\exists\vec{x}(R(\vec{x},\vec{x}')|_{\upsilon=0}\wedge S(\vec{x})|_{\upsilon=0})) & \upsilon\in X' \end{cases}$$

Beweis.

$$\exists\vec{x}[R(\vec{x},\vec{x}')\wedge S(\vec{x})] \overset{Shan.}{\underset{Exp.}{=}} \exists\vec{x}(\upsilon\cdot(R(\vec{x},\vec{x}')|_{\upsilon=1}\wedge S(\vec{x})|_{\upsilon=1})\vee\overline{\upsilon}\cdot(R(\vec{x},\vec{x}')|_{\upsilon=0}\wedge S(\vec{x})|_{\upsilon=0}))$$

$$\overset{\exists\vec{x}}{\underset{*}{=}} ((\exists\vec{x}(\upsilon\cdot(R(\vec{x},\vec{x}')|_{\upsilon=1}\wedge S(\vec{x})|_{\upsilon=1})))\vee(\exists\vec{x}(\overline{\upsilon}\cdot(R(\vec{x},\vec{x}')|_{\upsilon=0}\wedge S(\vec{x})|_{\upsilon=0}))))$$

$$\overset{\exists\vec{x}}{\underset{*}{=}} \begin{cases} (\exists_\upsilon(\upsilon\cdot(\exists(\vec{x}\ /\ \upsilon)(R(\vec{x},\vec{x}')|_{\upsilon=1}\wedge S(\vec{x})|_{\upsilon=1}))))\vee & \\ (\exists_\upsilon(\overline{\upsilon}\cdot(\exists(\vec{x}\ /\ \upsilon)(R(\vec{x},\vec{x}')|_{\upsilon=0}\wedge S(\vec{x})|_{\upsilon=0})))) & \upsilon\in X \\ & \\ \upsilon\cdot(\exists\vec{x}(R(\vec{x},\vec{x}')|_{\upsilon=1}\wedge S(\vec{x})|_{\upsilon=1}))\vee & \\ \overline{\upsilon}\cdot(\exists\vec{x}(R(\vec{x},\vec{x}')|_{\upsilon=0}\wedge S(\vec{x})|_{\upsilon=0})) & \upsilon\in X' \end{cases}$$

$$\overset{\exists\vec{x}}{\underset{*}{=}} \begin{cases} (\upsilon\cdot(\exists(\vec{x}\ /\ \upsilon)(R(\vec{x},\vec{x}')|_{\upsilon=1}\wedge S(\vec{x})|_{\upsilon=1})))|_{\upsilon=1}\vee & \\ (\upsilon\cdot(\exists(\vec{x}\ /\ \upsilon)(R(\vec{x},\vec{x}')|_{\upsilon=1}\wedge S(\vec{x})|_{\upsilon=1})))|_{\upsilon=0}\vee & \\ (\overline{\upsilon}\cdot(\exists(\vec{x}\ /\ \upsilon)(R(\vec{x},\vec{x}')|_{\upsilon=0}\wedge S(\vec{x})|_{\upsilon=0})))|_{\upsilon=1}\vee & \upsilon\in X \\ (\overline{\upsilon}\cdot(\exists(\vec{x}\ /\ \upsilon)(R(\vec{x},\vec{x}')|_{\upsilon=0}\wedge S(\vec{x})|_{\upsilon=0})))|_{\upsilon=0} & \\ & \\ \upsilon\cdot(\exists\vec{x}(R(\vec{x},\vec{x}')|_{\upsilon=1}\wedge S(\vec{x})|_{\upsilon=1}))\vee & \\ \overline{\upsilon}\cdot(\exists\vec{x}(R(\vec{x},\vec{x}')|_{\upsilon=0}\wedge S(\vec{x})|_{\upsilon=0})) & \upsilon\in X' \end{cases}$$

$$
\begin{array}{l}
\underset{*}{=} \begin{cases}
(\upsilon|_{\upsilon=1} \cdot (\exists\vec{x}(R(\vec{x},\vec{x}')|_{\upsilon=1} \wedge S(\vec{x})|_{\upsilon=1}))|_{\upsilon=1}) \vee \\
(\upsilon|_{\upsilon=0} \cdot (\exists\vec{x}(R(\vec{x},\vec{x}')|_{\upsilon=1} \wedge S(\vec{x})|_{\upsilon=1}))|_{\upsilon=0}) \vee \\
(\overline{\upsilon}|_{\upsilon=1} \cdot (\exists\vec{x}(R(\vec{x},\vec{x}')|_{\upsilon=0} \wedge S(\vec{x})|_{\upsilon=0}))|_{\upsilon=1}) \vee \\
(\overline{\upsilon}|_{\upsilon=0} \cdot (\exists\vec{x}(R(\vec{x},\vec{x}')|_{\upsilon=0} \wedge S(\vec{x})|_{\upsilon=0}))|_{\upsilon=0}) & \upsilon \in X \\
\\
\upsilon \cdot (\exists\vec{x}(R(\vec{x},\vec{x}')|_{\upsilon=1} \wedge S(\vec{x})|_{\upsilon=1})) \vee \\
\overline{\upsilon} \cdot (\exists\vec{x}(R(\vec{x},\vec{x}')|_{\upsilon=0} \wedge S(\vec{x})|_{\upsilon=0})) & \upsilon \in X'
\end{cases} \\
\\
\underset{*}{=} \begin{cases}
1 \cdot (\exists\vec{x}(R(\vec{x},\vec{x}')|_{\upsilon=1} \wedge S(\vec{x})|_{\upsilon=1})) \vee \\
0 \cdot (\exists\vec{x}(R(\vec{x},\vec{x}')|_{\upsilon=1} \wedge S(\vec{x})|_{\upsilon=1})) \vee \\
0 \cdot (\exists\vec{x}(R(\vec{x},\vec{x}')|_{\upsilon=0} \wedge S(\vec{x})|_{\upsilon=0})) \vee \\
1 \cdot (\exists\vec{x}(R(\vec{x},\vec{x}')|_{\upsilon=0} \wedge S(\vec{x})|_{\upsilon=0})) & \upsilon \in X \\
\\
\upsilon \cdot (\exists\vec{x}(R(\vec{x},\vec{x}')|_{\upsilon=1} \wedge S(\vec{x})|_{\upsilon=1})) \vee \\
\overline{\upsilon} \cdot (\exists\vec{x}(R(\vec{x},\vec{x}')|_{\upsilon=0} \wedge S(\vec{x})|_{\upsilon=0})) & \upsilon \in X'
\end{cases} \\
\\
= \begin{cases}
\exists\vec{x}(R(\vec{x},\vec{x}')|_{\upsilon=1} \wedge S(\vec{x})|_{\upsilon=1}) \vee \\
\exists\vec{x}(R(\vec{x},\vec{x}')|_{\upsilon=0} \wedge S(\vec{x})|_{\upsilon=0}) & \upsilon \in X \\
\\
\upsilon \cdot (\exists\vec{x}(R(\vec{x},\vec{x}')|_{\upsilon=1} \wedge S(\vec{x})|_{\upsilon=1})) \vee \\
\overline{\upsilon} \cdot (\exists\vec{x}(R(\vec{x},\vec{x}')|_{\upsilon=0} \wedge S(\vec{x})|_{\upsilon=0})) & \upsilon \in X'
\end{cases}
\end{array}
$$

Die mit * gekennzeichneten Rechenschritte werden in im weiteren Verlauf dieser Arbeit aufgeführten anderen Beweisen referenziert und bei diesen bei ausführlicherem Beweis entsprechend ausgeführt. In referenzierenden Beweisen wird zur Vereinfachung nur das entsprechende Ergebnis des letzten mit * gekennzeichneten Rechenschritts angegeben. □

Im Beweis zu Lemma 3.1 wurde gezeigt, dass die existentielle Quantifizierung bei Aufspaltung der Berechnung $\exists\vec{x}[R(\vec{x},\vec{x}') \wedge S(\vec{x})]$ in Teilprobleme bezüglich einer Booleschen Variablen, separat für die sich ergebenden Teilprobleme durchgeführt werden kann. Außerdem ist zu sehen, dass bei der Aufspaltung für eine Variable für aktuelle Zustände das Ergebnis der Anwendung der für die Variable für aktuelle Zustände auszuführenden existentiellen Quantifizierung, die Veroderung der Resultate der beiden Teilprobleme ist. Beim relationalen Produkt ist neben der UND-Verknüpfung und der existentiellen Quantifizierung bezüglich der Variablen für aktuelle Zustände, noch die Umbenennung von Variablen für Nachfolgerzustände in deren korrespondierende Variablen für aktuelle Zustände auszuführen. Für die Umbenennungsoperation und charakteristische Funktionen, die nur von der Menge der umzubenennenden Variablen abhängen, wurde die Distributivität bezüglich der Disjunktion und der Konjunktion in Abschnitt 2.5.2 in Lemma 2.42 gezeigt. Nach Durchführung der im relationalen Produkt enthaltenen existentiellen Quantifizierung für Teilergebnisse, sind im Resultat nur noch umzubenennende Variablen für Nachfolgerzustände wesentliche Variablen (siehe Abschnitt 2.2.1).

In Satz 3.2 werden Aufspaltungen der im relationalen Produkt durchzuführenden Berechnung in Teilprobleme, aus sich durch Belegung einer Booleschen Variablen mit Werten ergebenden Kofaktoren angegeben. Da die existentielle Quantifizierung in Satz 3.2 bei der Aufspaltung in Teilprobleme immer vor der Umbenennungsoperation berechnet wird (siehe Abschnitt 2.5.1), kann Lemma 2.42 hier angewendet werden.

Satz 3.2. *Gegeben sei das relationale Produkt $\exists\vec{x}[R(\vec{x},\vec{x}') \wedge S(\vec{x})]\{\vec{x}' \leftarrow \vec{x}\}$, wie es in Definition 2.35 eingeführt wurde. Sei außerdem v eine Boolesche Variable aus einer der dort definierten Variablenmengen X und X'. Dann kann die Berechnung des relationalen Produkts $\exists\vec{x}[R(\vec{x},\vec{x}') \wedge S(\vec{x})]\{\vec{x}' \leftarrow \vec{x}\}$ in Abhängigkeit davon, ob v eine Variable aus X oder X' ist, folgendermaßen aufgespalten werden:*

$$\exists\vec{x}[R(\vec{x},\vec{x}') \wedge S(\vec{x})]\{\vec{x}' \leftarrow \vec{x}\} = \begin{cases} (\exists\vec{x}(R(\vec{x},\vec{x}')|_{v=1} \wedge S(\vec{x})|_{v=1}))\{\vec{x}' \leftarrow \vec{x}\} \vee & v \in X \\ (\exists\vec{x}(R(\vec{x},\vec{x}')|_{v=0} \wedge S(\vec{x})|_{v=0}))\{\vec{x}' \leftarrow \vec{x}\} & \\ & \\ (v)\{\vec{x}' \leftarrow \vec{x}\} \cdot (\exists\vec{x}(R(\vec{x},\vec{x}')|_{v=1} \wedge S(\vec{x})|_{v=1})) & \\ \{\vec{x}' \leftarrow \vec{x}\} \vee (\overline{v})\{\vec{x}' \leftarrow \vec{x}\} \cdot (\exists\vec{x}(R(\vec{x},\vec{x}')|_{v=0} & v \in X' \\ \wedge S(\vec{x})|_{v=0}))\{\vec{x}' \leftarrow \vec{x}\} & \end{cases}$$

Beweis.

$$\exists\vec{x}[R(\vec{x},\vec{x}') \wedge S(\vec{x})]\{\vec{x}' \leftarrow \vec{x}\}$$

$$\overset{Lemma}{\underset{3.1}{=}} \begin{cases} [(\exists\vec{x}(R(\vec{x},\vec{x}')|_{v=1} \wedge S(\vec{x})|_{v=1})) \vee (\exists\vec{x} & v \in X \\ (R(\vec{x},\vec{x}')|_{v=0} \wedge S(\vec{x})|_{v=0}))]\{\vec{x}' \leftarrow \vec{x}\} & \\ & \\ [v \cdot (\exists\vec{x}(R(\vec{x},\vec{x}')|_{v=1} \wedge S(\vec{x})|_{v=1})) \vee \overline{v}\cdot & \\ (\exists\vec{x}(R(\vec{x},\vec{x}')|_{v=0} \wedge S(\vec{x})|_{v=0}))]\{\vec{x}' \leftarrow \vec{x}\} & v \in X' \end{cases}$$

$$\overset{Lemma}{\underset{2.42}{=}} \begin{cases} (\exists\vec{x}(R(\vec{x},\vec{x}')|_{v=1} \wedge S(\vec{x})|_{v=1}))\{\vec{x}' \leftarrow \vec{x}\} \vee & v \in X \\ (\exists\vec{x}(R(\vec{x},\vec{x}')|_{v=0} \wedge S(\vec{x})|_{v=0}))\{\vec{x}' \leftarrow \vec{x}\} & \\ & \\ (v)\{\vec{x}' \leftarrow \vec{x}\} \cdot (\exists\vec{x}(R(\vec{x},\vec{x}')|_{v=1} \wedge S(\vec{x})|_{v=1})) & \\ \{\vec{x}' \leftarrow \vec{x}\} \vee (\overline{v})\{\vec{x}' \leftarrow \vec{x}\} \cdot (\exists\vec{x}(R(\vec{x},\vec{x}')|_{v=0} & v \in X' \\ \wedge S(\vec{x})|_{v=0}))\{\vec{x}' \leftarrow \vec{x}\} & \end{cases}$$

□

Die Aufspaltung in Teilprobleme und die Kombination der resultierenden Teilergebnisse erfolgt in Algorithmus 9 wie in Satz 3.2 angegeben. In Algorithmus 9 werden diese Aufspaltungen in Teilprobleme rekursiv und jeweils bezüglich der für einen Rekursionsschritt ermittelten Top-Variablen angewandt. Falls die beiden ermittelten Teilergebnisse

gleich sind, führen beide Fälle von Satz 3.2 zu diesem Teilergebnis als Rückgabewert. Dieser Fall ist im Algorithmus in den Zeilen 11 und 12 implementiert. Nach der in Satz 3.2 angegebenen Berechnungsvorschrift, ist das Ergebnis eines Rekursionsschritts mit einer Top-Variablen v zur Kodierung von aktuellen Zuständen ($v \in X$) die Veroderung der ermittelten Teilergebnisse. Im Algorithmus ist die für diesen Fall durchzuführende Veroderung in Zeile 15 zu finden. Für eine Top-Variable v zur Kodierung von Nachfolgerzuständen ($v \in X'$) ist das Ergebnis eines solchen Rekursionsschritts im Satz ein neuer Knoten, für die dieser Variablen entsprechende Variable für aktuelle Zustände ($(v)\{\vec{x}' \leftarrow \vec{x}\}$ und $(\overline{v})\{\vec{x}' \leftarrow \vec{x}\}$). Dieser besitzt als Söhne die entsprechenden ermittelten Teilergebnisse. Im Algorithmus findet die Berechnung des Rückgabewerts nach dieser Berechnungsvorschrift in Zeile 17 statt. Damit entspricht das vom Algorithmus rekursiv angewendete Vorgehen, der in Satz 3.2 für die Aufspaltung der Berechnung eines Rekursionsschritts in Teilprobleme als korrekt bewiesenen Vorgehensweise.

Wichtig für die effiziente Anwendbarkeit dieses Vorgehens ist es, dass die zur Repräsentation von charakteristischen Funktionen verwendete Datenstruktur die effektive Ausführung der dazu benötigten Operationen, bei Verwendung von BDDs vor allem auch der Umbenennungsoperation, erlaubt. Dies ist bei BDDs und verschachtelten Variablenordnungen, für die der in Abschnitt 3.1 eingeführte Algorithmus entwickelt wurde, der Fall.

3.3 Korrektheit der neuen Terminalfälle

In diesem Abschnitt wird die Korrektheit der in Algorithmus 9 neu eingeführten Terminalfälle, welche in Algorithmus 10 aufgeführt sind, gezeigt. Um Vorteile durch die Benutzung der neuen Terminalfälle zu erhalten, ist es wichtig, dass die zur Erkennung der Terminalfälle nötigen Operationen und die Berechnung des Rückgabewerts der Terminalfälle von der verwendeten Datenstruktur effizient unterstützt werden. Für BDDs und die in Algorithmus 9 neu verwendeten Terminalfälle sind die dazu benötigten Operationen, Vergleich auf äquivalente repräsentierte Funktion bzw. Test auf die Repräsentation einer konstanten Funktion und Rückgabe einer konstanten Funktion, effizient in konstantem Aufwand durchführbar. Die Bedingungen für das Eintreten der neu verwendeten Terminalfälle sind in Zeile 4 in Algorithmus 10 zu sehen. Bei beiden neuen Terminalfällen ist der Rückgabewert die konstante Funktion mit dem Funktionswert 1.

Die Korrektheit der neuen Terminalfälle wird anschließend in den Beweisen zu Satz 3.3 und Satz 3.4 gezeigt. In diesen Sätzen werden jeweils zwei Fälle für das Resultat des relationalen Produkts unterschieden und bewiesen. Die in den Sätzen bewiesenen Fälle, die eintreten wenn keine der charakteristischen Funktionen $R(\vec{x}, \vec{x}')$ und $S(\vec{x})$ die konstante Funktion für den Booleschen Wert 0 ist, entsprechen den neu eingeführten Terminalfällen. Ist eine dieser charakteristischen Funktionen die konstante Funktion für den Wert 0, sind vorher bereits die Terminalfälle in Zeile 2 von Algorithmus 10 erfüllt und es wird vom Algorithmus die konstante Funktion für den Booleschen Wert 0 als Rückgabewert zurückgegeben. Im Folgenden wird im Beweis zu Satz 3.3 zuerst die Korrektheit des Terminalfalls, in dem die charakteristische Funktion für eine Menge von Transitionen der

konstanten Funktion mit dem Funktionswert 1 entspricht (Abfrage *IsTerminalVertexOne(R)* in Algorithmus 10), gezeigt.

Satz 3.3. *Gegeben sei das relationale Produkt* $\exists\vec{x}[R(\vec{x},\vec{x}') \wedge S(\vec{x})]\{\vec{x}' \leftarrow \vec{x}\}$*, wie es in Definition 2.35 eingeführt wurde. Ist* $R(\vec{x},\vec{x}') = 1\ \forall(x_1, x_2, ..., x_n, x'_1, x'_2, ..., x'_n) \in \{0,1\}^{2n}$*, dann gilt*

- $\exists\vec{x}[R(\vec{x},\vec{x}') \wedge S(\vec{x})]\{\vec{x}' \leftarrow \vec{x}\} = 0$*, falls* $S(\vec{x}) = 0\ \forall(x_1, x_2, ..., x_n) \in \{0,1\}^n$*.*
- $\exists\vec{x}[R(\vec{x},\vec{x}') \wedge S(\vec{x})]\{\vec{x}' \leftarrow \vec{x}\} = 1$*, falls* $\exists(x_1, x_2, ..., x_n) \in \{0,1\}^n$ *mit* $S(\vec{x}) = 1$*.*

Beweis. Gilt $R(\vec{x},\vec{x}') = 1\ \forall(x_1, x_2, ..., x_n, x'_1, x'_2, ..., x'_n) \in \{0,1\}^{2n}$, dann lassen sich die folgenden zwei Fälle unterscheiden:

Fall 1: $S(\vec{x}) = 0\ \forall(x_1, x_2, ..., x_n) \in \{0,1\}^n$

$$\exists\vec{x}[R(\vec{x},\vec{x}') \wedge S(\vec{x})]\{\vec{x}' \leftarrow \vec{x}\} = \exists\vec{x}[1 \wedge 0]\{\vec{x}' \leftarrow \vec{x}\} = \exists\vec{x}[0]\{\vec{x}' \leftarrow \vec{x}\} = 0$$

Fall 2: $\exists(x_1, x_2, ..., x_n) \in \{0,1\}^n$ mit $S(\vec{x}) = 1$

$$\begin{aligned}
\exists\vec{x}[R(\vec{x},\vec{x}') \wedge S(\vec{x})]\{\vec{x}' \leftarrow \vec{x}\} = \exists\vec{x}[1 \wedge S(\vec{x})]\{\vec{x}' \leftarrow \vec{x}\} &= \exists\vec{x}[S(\vec{x})]\{\vec{x}' \leftarrow \vec{x}\} \\
&\overset{\substack{Shan.\\Exp.}}{=} \exists\vec{x}[\overline{x_1}\,\overline{x_2} \ldots \overline{x_n} S(\vec{x})|_{x_1=0,\ldots,x_n=0} \\
&\vee x_1\overline{x_2} \ldots \overline{x_n} S(\vec{x})|_{x_1=1,\,x_2=0,\ldots,x_n=0} \vee \ldots \\
&\vee x_1 x_2 \ldots x_n S(\vec{x})|_{x_1=1,\ldots,x_n=1}]\{\vec{x}' \leftarrow \vec{x}\} \\
&\overset{\exists\vec{x}}{=} [S(\vec{x})|_{x_1=0,\ldots,x_n=0} \vee S(\vec{x})|_{x_1=1,\,x_2=0,\ldots,x_n=0} \vee \ldots \\
&\vee S(\vec{x})|_{x_1=1,\ldots,x_n=1}]\{\vec{x}' \leftarrow \vec{x}\} \\
&= [1]\{\vec{x}' \leftarrow \vec{x}\} = 1
\end{aligned}$$

□

Die Korrektheit des zweiten neuen Terminalfalls wird anschließend im Beweis zu Satz 3.4 gezeigt. Bei diesem Terminalfall entspricht die charakteristische Funktion für eine Menge von Transitionen der charakteristischen Funktion der Menge der zu explorierenden Zustände (Abfrage $R{==}S$ in Algorithmus 10).

Satz 3.4. *Gegeben sei das relationale Produkt* $\exists\vec{x}[R(\vec{x},\vec{x}') \wedge S(\vec{x})]\{\vec{x}' \leftarrow \vec{x}\}$*, wie es in Definition 2.35 eingeführt wurde. Ist* $R(\vec{x},\vec{x}') = S(\vec{x})\ \forall(x_1, x_2, ..., x_n, x'_1, x'_2, ..., x'_n) \in \{0,1\}^{2n}$*, dann gilt*

- $\exists\vec{x}[R(\vec{x},\vec{x}') \wedge S(\vec{x})]\{\vec{x}' \leftarrow \vec{x}\} = 0$*, falls* $R(\vec{x},\vec{x}') = 0\ \forall(x_1, x_2, ..., x_n, x'_1, x'_2, ..., x'_n) \in \{0,1\}^{2n}$*.*
- $\exists\vec{x}[R(\vec{x},\vec{x}') \wedge S(\vec{x})]\{\vec{x}' \leftarrow \vec{x}\} = 1$*, falls* $\exists(x_1, x_2, ..., x_n, x'_1, x'_2, ..., x'_n) \in \{0,1\}^{2n}$ *mit* $R(\vec{x},\vec{x}') = 1$*.*

Beweis. Gilt $R(\vec{x},\vec{x}') = S(\vec{x})\ \forall(x_1,x_2,...,x_n,x'_1,x'_2,...,x'_n) \in \{0,1\}^{2n}$, dann lassen sich die folgenden zwei Fälle unterscheiden:

Fall 1: $R(\vec{x},\vec{x}') = 0\ \forall(x_1,x_2,...,x_n,x'_1,x'_2,...,x'_n) \in \{0,1\}^{2n}$

$$\exists\vec{x}[R(\vec{x},\vec{x}') \wedge S(\vec{x})]\{\vec{x}' \leftarrow \vec{x}\} = \exists\vec{x}[0 \wedge 0]\{\vec{x}' \leftarrow \vec{x}\} = \exists\vec{x}[0]\{\vec{x}' \leftarrow \vec{x}\} = 0$$

Fall 2: $\exists(x_1,x_2,...,x_n,x'_1,x'_2,...,x'_n) \in \{0,1\}^{2n}$ mit $R(\vec{x},\vec{x}') = 1$

Da $R(\vec{x},\vec{x}') = S(\vec{x})\ \forall(x_1,x_2,...,x_n,x'_1,x'_2,...,x'_n) \in \{0,1\}^{2n}$, besitzt $R(\vec{x},\vec{x}')$ nur wesentliche Variablen aus der Menge X und $\exists(x_1,x_2,...,x_n,x'_1,x'_2,...,x'_n) \in \{0,1\}^{2n}$ mit $R(\vec{x},\vec{x}') = S(\vec{x}) = 1$. Damit gilt:

$$\begin{aligned}
\exists\vec{x}[R(\vec{x},\vec{x}') \wedge S(\vec{x})]\{\vec{x}' \leftarrow \vec{x}\} &= \exists\vec{x}[R(\vec{x},\vec{x}')]\{\vec{x}' \leftarrow \vec{x}\} \\
&\overset{\substack{Shan.\\Exp.}}{=} \exists\vec{x}[\overline{x_1}\,\overline{x_2}\,...\,\overline{x_n}R(\vec{x},\vec{x}')|_{x_1=0,...,x_n=0} \\
&\vee x_1\overline{x_2}\,...\,\overline{x_n}R(\vec{x},\vec{x}')|_{x_1=1,x_2=0,...,x_n=0} \vee ... \\
&\vee x_1x_2\,...\,x_nR(\vec{x},\vec{x}')|_{x_1=1,...,x_n=1}]\{\vec{x}' \leftarrow \vec{x}\} \\
&\overset{\exists\vec{x}}{=} [R(\vec{x},\vec{x}')|_{x_1=0,...,x_n=0} \vee R(\vec{x},\vec{x}')|_{x_1=1,x_2=0,...,x_n=0} \vee ... \\
&\vee R(\vec{x},\vec{x}')|_{x_1=1,...,x_n=1}]\{\vec{x}' \leftarrow \vec{x}\} \\
&= [1]\{\vec{x}' \leftarrow \vec{x}\} = 1
\end{aligned}$$

□

Damit wurde die Korrektheit der beiden neu eingeführten Terminalfälle gezeigt. Durch Verwendung dieser Terminalfälle kann das weitere Durchlaufen der BDDs vom Algorithmus zur kombinierten Berechnung des relationalen Produkts früher gestoppt werden, als bei Benutzung der im Algorithmus aus [8] verwendeten Terminalfälle. Daraus können sich Laufzeitverbesserungen und auch eine Reduktion des Speicherbedarfs ergeben.

3.4 Relationales Produkt für Vorgängerberechnungen

Neben der Vorwärtserreichbarkeitsanalyse, bei der wiederholt Nachfolgerberechnungen zur Ermittlung aller von einer Zustandsmenge erreichbaren Zustände durchgeführt werden, wurde in Abschnitt 2.1.6.3 auch die Rückwärtserreichbarkeitsanalyse vorgestellt. Bei dieser werden durch wiederholte Vorgängerberechnungen alle Vorgängerzustände einer gegebenen Zustandsmenge berechnet.

Der in Abschnitt 3.1 präsentierte Algorithmus berechnet das relationale Produkt für die Nachfolgerberechnung (siehe Definition 2.35 in Abschnitt 2.5.1) zur Ermittlung der unmittelbaren Nachfolgerzustände einer Zustandsmenge. In diesem Abschnitt wird beschrieben, wie dieser Algorithmus zur Verwendung für Vorgängerberechnungen angepasst

werden kann. Die Berechnung einer Menge von Vorgängerzuständen einer Zustandsmenge kann durch das relationale Produkt für die Vorgängerberechnung beschrieben werden.

Definition 3.5. (Relationales Produkt für die Vorgängerberechnung) *Seien* $X = \{x_1, x_2, ..., x_n\}$ *und* $X' = \{x'_1, x'_2, ..., x'_n\}$ *Mengen von Booleschen Variablen,*
$\vec{x} = (x_1, x_2, ..., x_n)$ *bzw.* $\vec{x}' = (x'_1, x'_2, ..., x'_n)$ *Vektoren über diesen Booleschen Variablen und* $R(\vec{x}, \vec{x}')$ *und* $S(\vec{x}')$ *Boolesche charakteristische Funktionen. Dann heißt der Ausdruck*

$$\exists \vec{x}'[R(\vec{x}, \vec{x}') \wedge S(\vec{x}')]\{\vec{x} \leftarrow \vec{x}'\} \tag{3.1}$$

relationales Produkt für die Vorgängerberechnung.

Im relationalen Produkt für die Vorgängerberechnung ist $R(\vec{x}, \vec{x}')$ die charakteristische Funktion einer Menge von Transitionen und $S(\vec{x}')$ die charakteristische Funktion der Zustandsmenge, für die Vorgängerzustände zu berechnen sind. In der durch R repräsentierten Menge von Transitionen sind die durch diese Transitionen gegebenen Zuordnungen von Zuständen, für die Vorgängerzustände zu berechnen sind (repräsentiert über Variablen aus X'), zu Vorgängerzuständen (repräsentiert über Variablen aus X) abgespeichert. Eine Menge von Vorgängerzuständen $S'(\vec{x}')$, die ebenfalls über Variablen der Menge X' repräsentiert wird, kann damit mit dem relationalen Produkt für die Vorgängerberechnung gemäß $S'(\vec{x}') = \exists \vec{x}'[R(\vec{x}, \vec{x}') \wedge S(\vec{x}')]\{\vec{x} \leftarrow \vec{x}'\}$ berechnet werden. Für die Vorgängerberechnung und die Variablenmengen X' und X werden nachfolgend die Bezeichnungen *Variablen für aktuelle Zustände* (in Definition 3.6) und *Variablen für Vorgängerzustände* (in Definition 3.7) eingeführt.

Definition 3.6. (Variablen für aktuelle Zustände) *Die im relationalen Produkt für die Vorgängerberechnung* $\exists \vec{x}'[R(\vec{x}, \vec{x}') \wedge S(\vec{x}')]\{\vec{x} \leftarrow \vec{x}'\}$ *vorkommenden Variablen der Menge* $X' = \{x'_1, x'_2, ..., x'_n\}$ *heißen Variablen für aktuelle Zustände.*

Definition 3.7. (Variablen für Vorgängerzustände) *Die im relationalen Produkt für die Vorgängerberechnung* $\exists \vec{x}'[R(\vec{x}, \vec{x}') \wedge S(\vec{x}')]\{\vec{x} \leftarrow \vec{x}'\}$ *vorkommenden Variablen der Menge* $X = \{x_1, x_2, ..., x_n\}$ *heißen Variablen für Vorgängerzustände.*

Beim relationalen Produkt für die Nachfolgerberechnung wurden die Variablen der Menge X als Variablen für aktuelle Zustände eingeführt (siehe Abschnitt 2.5.1). Hier werden die Variablen der Menge X' bezüglich des relationalen Produkts für die Vorgängerberechnung als Variablen für aktuelle Zustände bezeichnet. In beiden Fällen bezieht sich der Begriff aktuelle Zustände auf die zu explorierenden Zustände, mit denen die Transitionsrelation bei Bildberechnungen kombiniert wird. Im Gegensatz zur Nachfolgerberechnung erfolgt die existentielle Quantifizierung im relationalen Produkt für die Vorgängerberechnung nicht bezüglich der Variablen aus der Menge X ($\exists \vec{x}$), sondern bezüglich der Variablen der Menge X' ($\exists \vec{x}'$). Zusätzlich ist die Umbenennungsoperation in umgekehrter Richtung, nämlich durch Umbenennung der Variablen für Vorgängerzustände in die zu diesen korrespondierenden Variablen für aktuelle Zustände, durchzuführen $\{\vec{x} \leftarrow \vec{x}'\}$.

Um Algorithmus 9 für die Berechnung des relationalen Produkts für die Vorgängerberechnung anzupassen, sind lediglich die Zeilen $11 - 17$ des Algorithmus und die dort stattfindende Kombination von Teilergebnissen zum Rückgabewert eines Rekursionsschritts

abzuändern. Dort werden die im relationalen Produkt enthaltene existentielle Quantifizierung und die Umbenennungsoperation ausgeführt. Die bisher verwendeten Terminalfälle, die Benutzung der Ergebnistabelle (bis auf die Anpassung der Operationsart), die Auswahl der Top-Variable eines Rekursionsschritts und die rekursiven Aufrufe des Algorithmus können unverändert beibehalten werden.

Eine für die Berechnung des relationalen Produkts für die Vorgängerberechnung angepasste Berechnung des Resultats eines Rekursionsschritts ist in Algorithmus 11 dargestellt. Sind die beiden BDDs R_0 bzw. R_1 der rekursiv ermittelten Teilergebnisse gleich,

Algorithmus 11: Für die Berechnung des relationalen Produkts für die Vorgängerberechnung angepasste Resultatsberechnung von Algorithmus 9.

```
1 if R0 == R1 then
2 |  return R1;
3 else
4 |  if IsPredecessorStateVar(v) then
5 |  |  Result = FindOrAddUniqueTable(GetCurrentStateVarPred(v), R0, R1);
6 |  else
7 |  |  Result = APPLY(Or, R0, R1);
```

so ist bei der Vorgängerberechnung wie bei der Nachfolgerberechnung aufgrund der Reduktionsregeln für BDDs, kein neuer Knoten für den Rückgabewert des aktuellen Rekursionsschritts zu erzeugen. Beide Belegungen der Top-Variable führen beim Ergebnis des Rekursionsschritts zum gleichen BDD für ein Teilresultat. Andernfalls unterscheidet sich die Resultatsberechnung vom Algorithmus für die Nachfolgerberechnung. Ist die aktuelle Top-Variable v bei der Vorgängerberechnung eine Variable für Vorgängerzustände ($v \in X$ und *IsPredecessorStateVar(v)* gibt den Wert *TRUE* zurück), so ist für diese die Umbenennung in die entsprechende Variable für aktuelle Zustände ($v \in X'$) durchzuführen. Dazu wird in Zeile 5 von Algorithmus 11 ein neuer Knoten für diese Variable für aktuelle Zustände erzeugt. Die zur Variablen v korrespondierende Variable für aktuelle Zustände wird im Algorithmus durch die Funktion *GetCurrentStateVarPred()* berechnet. Ist die aktuelle Top-Variable v bei der Vorgängerberechnung eine Variable für aktuelle Zustände ($v \in X'$), wird in Zeile 7 des Algorithmus die für Variablen für aktuelle Zustände notwendige existentielle Quantifizierung durchgeführt. Dazu ist die Veroderung der beiden Teilresultate R_0 und R_1 durchzuführen, was wieder durch Aufruf der Funktion *APPLY* für die ODER-Verknüpfung gemacht wird.

3.5 Abgrenzung zu verwandten Arbeiten

In diesem Kapitel wurde für verschachtelte Variablenordnungen ein Algorithmus zur kombinierten Berechnung des relationalen Produkts bei der BDD-basierten Modellprüfung präsentiert. Weitere Ansätze zur Optimierung von Bildberechnungen bei der BDD-basierten Modellprüfung wurden in den Abschnitten 2.5.1, 2.5.2, 2.5.6 und 2.5.7 dieser Arbeit vorge-

stellt. Beim naiven in Abschnitt 2.5.1 beschriebenen Ansatz zur Berechnung des relationalen Produkts findet eine separate und aufeinanderfolgende Berechnung der im relationalen Produkt enthaltenen Operationen statt. Dabei können große BDDs für Zwischenergebnisse auftreten, obwohl das Ergebnis des relationalen Produkts selbst möglicherweise mit einem deutlich kleineren BDD repräsentiert werden kann. Dies kann durch die kombinierte Berechnung des relationalen Produkts und die gemeinsame Berechnung der darin enthaltenen Operationen abgemildert werden. Durch frühzeitige Anwendung der existentiellen Quantifizierung wird dort die Menge der wesentlichen Variablen reduziert, was zu kleineren BDDs führt. Außerdem führt die kombinierte Berechnung des relationalen Produkts, wie auch die in Abschnitt 8.5.1.1 präsentierten experimentellen Ergebnisse bestätigen, zu großen Laufzeitverbesserungen. Ein Ansatz, bei dem nur die UND-Verknüpfung und die existentielle Quantifizierung für beliebige Variablenordnungen gemeinsam in einer BDD-Traversierung berechnet werden, wurde in [142] vorgestellt. Bei der in diesem Kapitel für verschachtelte Variablenordnungen präsentierten kombinierten Berechnung des relationalen Produkts, wird zusätzlich noch die Umbenennungsoperation in der einen BDD-Traversierung durchgeführt. Dadurch kann die dafür notwendige separate BDD-Traversierung vermieden und die für diese benötigte Laufzeit eingespart werden. Verschachtelte Variablenordnungen, für die dieser Algorithmus implementiert wurde, sind häufig effiziente Variablenordnungen zur Repräsentation von Transitionssystemen (siehe Abschnitt 2.5.3).

Ein dem in dieser Arbeit vorgestellten Algorithmus ähnlicher Algorithmus zur kombinierten Berechnung aller im relationalen Produkt enthaltenen Operationen, wurde in [8] präsentiert. Dieser Algorithmus wurde dort für die Berechnung von Vorgängerzuständen angegeben. Bei diesem werden Paare von korrespondierenden Variablen für Vorgängerzustände und aktuelle Zustände in Rekursionsschritten immer gemeinsam betrachtet. Im Unterschied dazu wird bei dem in diesem Kapitel vorgestellten Algorithmus die Aufspaltung in Teilprobleme in einem Rekursionsschritt nur bezüglich einer für den Rekursionsschritt ermittelten Top-Variablen durchgeführt. Dadurch werden für Variablen für aktuelle Zustände und für Variablen für Nachfolgerzustände separate rekursive Aufrufe ausgeführt. Zusätzlich werden beim Algorithmus aus diesem Kapitel im Vergleich mit dem Algorithmus aus [8] optimierte Terminalfälle der rekursiven Berechnung verwendet. In [8] ist außerdem kein Beweis der Korrektheit der Vorgehensweise des dort vorgestellten Algorithmus aufgeführt. Ebenso werden keine experimentellen Ergebnisse zur Evaluation der Leistungsfähigkeit des Algorithmus, wie sie in der vorliegenden Arbeit angegeben sind, präsentiert. Es konnten auch keine weiteren Arbeiten gefunden werden, in denen Ergebnisse von Verifikationsexperimenten zum Vergleich der naiven sequentiellen und der kombinierten Berechnung des relationalen Produkts präsentiert wurden.

Ein *Early Quantification* genanntes Verfahren zur Bildberechnung mit konjunktiv partitionierten Transitionsrelationen wurde in Abschnitt 2.5.2 beschrieben. Dort wird die existentielle Quantifizierung für Variablen, von denen noch zu bearbeitende Partitionen der Transitionsrelation nicht mehr abhängen, nach der Berechnung der UND-Verknüpfung mit Partitionen so früh wie möglich ausgeführt. Es findet dort aber keine kombinierte Berechnung der Operationen zur Bildberechnung statt. In Abschnitt 2.5.6 wurden weitere Ansätze zur effizienteren Durchführung von Bildberechnungen vorgestellt. Keiner dieser

Ansätze zielt auf eine verbesserte Implementierung der im relationalen Produkt enthaltenen Operationen ab. Ein Vorschlag, bei dem ebenfalls das Verfahren zur Bildberechnung direkt geändert wird, ist die Verwendung von Operatoren und Algorithmen, die auf der Bildung verallgemeinerter Kofaktoren basieren. Diese werden anstatt der im relationalen Produkt enthaltenen UND-Verknüpfung benutzt. Sie definieren Funktionswerte um und versuchen dadurch kleinere BDDs für Zwischenergebnisse zu erhalten. In Abschnitt 2.5.7 wurden Verfahren zur Optimierung der Zustandsraumdurchmusterung bei der Erreichbarkeitsanalyse präsentiert. Bei diesen werden Optimierungen bei der Auswahl der Menge der Zustände mit denen Bildberechnungen durchgeführt werden und bei der Auswahl von Partitionen von partitionierten Transitionsrelationen für Bildberechnungen durchgeführt. Es findet dabei keine Änderung der im relationalen Produkt durchzuführenden Operationen statt, sondern es wurden dort Optimierungen der zur Berechnung des relationalen Produkts benutzten BDDs und der Explorationsreihenfolge mit BDDs entwickelt.

4 TLEBDDs

Ein Grund für einen hohen Speicherbedarf bei der BDD-basierten Modellprüfung ist häufig der große Speicherbedarf für BDDs zur Repräsentation der Transitionsrelation. Da die Benutzung eines monolithischen BDDs zur Repräsentation der Transitionsrelation oft einen großen Speicherbedarf erfordert, wurde die Verwendung partitionierter Transitionsrelationen vorgeschlagen (siehe Abschnitt 2.5.2). Bei partitionierten Transitionsrelationen wird die Transitionsrelation in mehrere Teile aufgespalten, die häufig jeweils durch kleine BDDs repräsentiert werden können. Dennoch kann der Speicherbedarf partitionierter Transitionsrelationen immer noch sehr groß sein. Außerdem kann die Verwendung zu kleiner Partitionen in partitionierten Transitionsrelationen zu Laufzeitanstiegen bei der Modellprüfung führen.

In Abschnitt 2.3 wurden asynchrone nebenläufige Systeme und die Eigenschaft der Transitionslokalität für solche Systeme eingeführt. Bei asynchronen nebenläufigen Systemen mit Transitionslokalität sind die Transitionen jeder Komponente nur von den Werten von globalen Variablen und dem eigenen lokalen Zustand der Komponente abhängig. Eine Komponente eines solchen Systems besitzt keinen Lese- und keinen Schreibzugriff auf die lokalen Variablen der anderen Komponenten. In diesem Kapitel werden *Transitionslokalitätsausnutzende binäre Entscheidungsdiagramme (engl. transition locality exploiting BDDs, TLEBDDs)* eingeführt. Ein TLEBDD ist eine Datenstruktur, die durch Ausnutzen der Transitionslokalität die kompaktere Repräsentation der Transitionsrelation von asynchronen nebenläufigen Systemen mit Transitionslokalität ermöglicht. Durch ihre Verwendung kann die Transitionsrelation mit deutlich weniger Speicherbedarf als bei Benutzung von BDDs repräsentiert werden (siehe Verifikationsexperimente in Abschnitt 8.4). Außerdem kann ihre Benutzung im Vergleich mit einer partitionierten Transitionsrelation aus BDDs zu großen Laufzeitverbesserungen bei der Vorwärtserreichbarkeitsanalyse führen (siehe Verifikationsexperimente in Abschnitt 8.5.1).

4.1 Die Datenstruktur TLEBDD

Transitionslokalitätsausnutzende binäre Entscheidungsdiagramme (engl. transition locality exploiting BDDs, TLEBDDs) dienen zur Repräsentation der Transitionsrelation von asynchronen nebenläufigen Systemen mit Transitionslokalität. Bei diesen hängen Transitionen einer Komponente nur von den Werten der globalen Variablen und vom eigenen lokalen Zustand der Komponente ab. Werden Transitionen eines solchen Systems durch eine komponentenweise partitionierte Transitionsrelation (siehe Abschnitt 2.5.2) repräsentiert, können Teilgraphen von BDDs zur Repräsentation von Transitionen verschiedener Komponenten Ähnlichkeiten bei der Repräsentation der Veränderungen des globalen

Zustands und des lokalen Zustands von Komponenten aufweisen. Das Auftreten solcher Ähnlichkeiten ist abhängig von der für die BDDs verwendeten Variablenordnung und von der Partitionierung der Transitionen, da beides den Aufbau der BDDs beeinflusst. In Systemen mit replizierten Komponenten sind bei Verwendung einer komponentenweise partitionierten Transitionsrelation oft sehr viele strukturelle Ähnlichkeiten zwischen den BDDs zur Repräsentation der Transitionen verschiedener Komponenten vorhanden. Der Grund dafür sind bis auf mögliche Abweichungen bei den Wertänderungen von id-sensitiven Variablen (siehe Abschnitt 2.4.1.1) gleiche Verhaltensweisen der Komponenten, die durch die Transitionen der Komponenten festgelegt werden.

In BDDs werden die lokalen Zustände verschiedener Komponenten über unterschiedlichen Booleschen Variablen und damit durch Knoten für verschiedene Boolesche Variablen repräsentiert. Dadurch können ähnliche Teilgraphen zur Repräsentation der Veränderungen der lokalen Zustände verschiedener Komponenten von BDD-Programmpaketen nicht ausgenutzt werden, da sich diese Teilgraphen in den Variablenmarkierungen der Knoten unterscheiden. Zusätzlich sind aufgrund der Transitionslokalität in einem BDD zur Repräsentation von Transitionen einer Komponente möglicherweise viele Identitätstransformationen für die Zustandsänderungen des lokalen Zustands der anderen Komponenten durch Transitionen enthalten. Bei verschachtelten Variablenordnungen werden diese durch Identitätsmuster repräsentiert (siehe Abschnitt 2.5.3). TLEBDDs wurden entwickelt, um strukturelle Ähnlichkeiten zwischen BDDs zur Repräsentation der Transitionsrelation verschiedener Komponenten ausnutzen zu können und um den Speicherbedarf für Identitätstransformationen für lokale Zustandsänderungen anderer Komponenten zu vermeiden. Die in Abschnitt 8.4.1.2 zum Bau der Transitionsrelation mit TLEBDDs angegebenen Ergebnisse von Verifikationsexperimenten zeigen, dass damit der zur Repräsentation von Transitionsrelationen benötigte Speicherbedarf im Vergleich zur Benutzung von BDDs deutlich reduziert werden kann.

Mit einem TLEBDD können nur Transitionen einer Komponente repräsentiert werden. TLEBDDs können damit für komponentenweise partitionierte Repräsentationen von Transitionsrelationen verwendet werden. Ein TLEBDD besteht aus einem BDD und einer Zuordnungsliste. Der BDD eines TLEBDDs für Transitionen einer Komponente i ist über einer im Vergleich zur gewöhnlichen Repräsentation von Transitionen durch BDDs reduzierten Menge von Booleschen Variablen $Y = \{y_1, y_2, ..., y_{2\cdot(n_g+n_{l_i})}\}$ definiert. Dabei ist n_g hier die Anzahl der Booleschen Variablen zur Kodierung des globalen Zustands eines Systems und n_{l_i} ist die Anzahl der Booleschen Variablen zur Kodierung des lokalen Zustands der Komponente i (siehe Abschnitt 2.3). Die Variablen der Menge Y werden im Folgenden reduzierte Variablen genannt. Mit dem BDD eines TLEBDDs werden nur die Zustandsübergänge des globalen Zustands und die des lokalen Zustands der Komponente, für die der TLEBDD gebaut wurde, repräsentiert. Die Zuordnungsliste eines TLEBDDs beinhaltet dagegen Variablen der Variablenmenge $X = \{x_1, x_2, ..., x_{2\cdot N}\}$, die auch bei der Repräsentation von Transitionen durch gewöhnliche BDDs verwendet werden würde ($N = n_g + \sum_{i=1}^{m} n_{l_i}$, siehe Abschnitt 2.3). Die Variablen der Menge X werden im Folgenden auch tatsächliche Variablen genannt. Die Zuordnungsliste wird verwendet, um die den reduzierten Variablen eines TLEBDDs entsprechenden tatsächlichen Variablen eines

gewöhnlichen BDDs über der Variablenmenge X abzuspeichern. Dies ist für die Verknüpfung von mit TLEBDDs repräsentierten Transitionen mit durch BDDs repräsentierten Zustandsmengen erforderlich. Lokale Zustände verschiedener Komponenten werden bei Verwendung von BDDs über unterschiedlichen Booleschen Variablen repräsentiert. Daher unterscheiden sich die den reduzierten Variablen zur Kodierung des lokalen Zustands von Komponenten entsprechenden tatsächlichen Variablen zwischen Komponenten. Aus diesem Grund sind für die verschiedenen Komponenten unterschiedliche Zuordnungen der reduzierten Variablen zu den tatsächlichen Variablen nötig. Für eine Komponente i wird die Zuordnung der reduzierten Variablen zu den entsprechenden tatsächlichen Variablen durch eine Zuordnungsfunktion beschrieben.

Definition 4.1. (Zuordnungsfunktion) *Die Zuordnungsfunktion einer Komponente i eines asynchronen nebenläufigen Systems, die Variablenindizes einer Menge von reduzierten Variablen $Y = \{y_1, y_2, ..., y_{2\cdot(n_g+n_{l_i})}\}$ auf die diesen entsprechenden Variablenindizes der tatsächlichen Variablen $X = \{x_1, x_2, ..., x_{2\cdot N}\}$ abbildet, ist durch*

$$\pi_i : \{1, 2, .., 2 \cdot (n_g + n_{l_i})\} \rightarrow \{1, 2, .., 2 \cdot N\} \tag{4.1}$$

definiert.

Damit kann die Zuordnungsliste definiert werden.

Definition 4.2. (Zuordnungsliste) *Eine n-stellige Zuordnungsliste für eine Komponente i eines asynchronen nebenläufigen Systems mit $n = 2 \cdot (n_g + n_{l_i})$ ist eine Liste über Variablen der Menge X mit n-Elementen und folgender Form: $[x_{\pi_i(1)}, ..., x_{\pi_i(n)}]$.*

Wie in Abschnitt 2.3 beschrieben wurde, ist die Transitionsrelation R_i einer Komponente in einem System mit Transitionslokalität durch $R_i = \{((\vec{g}, l_1, ..., l_m), (\vec{g}', l'_1, ..., l'_m)) | \forall j \neq i : l'_j = l_j \wedge ((\vec{g}, l_i), (\vec{g}', l'_i)) \in R_{i_P}\}$ definiert. Die Relation R_{i_P} gibt dabei die Teile eines Zustands und eines unmittelbaren Nachfolgerzustands an, die eine Transition der Komponente i verändern kann. Bei vorhandener Transitionslokalität sind dies jeweils die globalen Variablen und die lokalen Variablen der Komponente i. Der lokale Zustand anderer Komponenten j, mit $j \in \{1, ..., m\}$ und $j \neq i$, ändert sich in Systemen mit Transitionslokalität durch Ausführen von Transitionen der Komponente i nicht. Würden BDDs zur Repräsentation der Transitionen einer Komponente i verwendet werden, würde das gleich bleiben des lokalen Zustands der anderen Komponenten durch Muster für Identitätstransformationen repräsentiert werden. Bei verschachtelten Variablenordnungen sind das die in Abschnitt 2.5.3 vorgestellten Identitätsmuster. Diese werden bei TLEBDDs nicht explizit abgespeichert.

Definition 4.3. (Transitionslokalitätsausnutzendes binäres Entscheidungsdiagramm (engl. transition locality exploiting BDD, TLEBDD)) *Gegeben sei ein asynchrones nebenläufiges System mit Transitionslokalität aus $m > 1$ Komponenten und eine binäre Kodierung von Zuständen und Transitionen des Systems über Variablen der Menge $X = \{x_1, x_2, ..., x_{2 \cdot N}\}$, mit $N = n_g + \sum_{i=1}^{m} n_{l_i}$, wie in Abschnitt 2.3 angegeben. Dann ist ein TLEBDD zur Repräsentation von Transitionen einer Komponente i eines solchen Systems ein Tupel (G, b) mit folgenden Eigenschaften:*

- *G ist ein BDD über der Variablenmenge $Y = \{y_1, y_2, ..., y_{2 \cdot (n_g + n_{l_i})}\}$, mit dem eine Teilmenge der Relation R_{i_P} der Komponente i repräsentiert wird. Die Variablen der Menge Y werden reduzierte Variablen genannt.*
- *b ist eine Zuordnungsliste, die für jede reduzierte Variable y_q mit $q \in \{1, 2, ..., 2 \cdot (n_g + n_{l_i})\}$ der Variablen des BDDs G die entsprechende tatsächliche Variable $x_{\pi_i(q)}$ beinhaltet. Dabei gilt $x_{\pi_i(q_1)} \neq x_{\pi_i(q_2)}$ für $q_1, q_2 \in \{1, 2, ..., 2 \cdot (n_g + n_{l_i})\}$ und $q_1 \neq q_2$.*
- *Für tatsächliche Variablen zur Kodierung der Änderungen der lokalen Zustände der anderen $m - 1$ neben i im System vorhandenen Komponenten werden implizit Identitätstransformationen angenommen.*

Durch die Entfernung von Identitätstransformationen und die dadurch mögliche Verwendung einer reduzierten Menge von Variablen, ist mit TLEBDDs bei Transitionslokalität eine kompaktere Darstellung der Transitionsrelation als mit partitionierten Transitionsrelationen aus BDDs möglich. Transitionen einer Komponente i können mit TLEBDDs durch Speicherung der Relation R_{i_P} oder einer Teilmenge dieser Relation im BDD des TLEBDDs, dessen Variablenmenge die Menge der reduzierten Variablen ist, repräsentiert werden. Zusätzlich muss in der Zuordnungsliste des TLEBDDs die Zuordnung der $2 \cdot (n_g + n_{l_i})$ reduzierten Variablen zu den $2 \cdot N$ korrespondierenden tatsächlichen Variablen, über denen ein entsprechender BDD definiert wäre, gespeichert werden.

Viele BDD-Programmpakete, wie auch das für die in dieser Arbeit durchgeführten Verifikationsexperimente benutzte BDD-Programmpaket CUDD [181], speichern isomorphe Teilgraphen (siehe Definition 2.21 für isomorphe binäre Entscheidungsdiagramme) nur einmal ab. Bei Benutzung von TLEBDDs zur Repräsentation der Transitionsrelation können Veränderungen des globalen Zustands und des lokalen Zustands durch Transitionen verschiedener Komponenten eines Systems über der gleichen Menge von reduzierten Variablen dargestellt werden. Dadurch wird es möglich strukturelle Ähnlichkeiten in den BDDs der TLEBDDs zur Kodierung der Relationen R_{i_P} verschiedener Komponenten auszunutzen. Aus dieser Speicherung über den gleichen Variablen resultierende isomorphe Teilgraphen, müssen von vielen BDD-Programmpaketen nur einmal abgespeichert werden. Bei gewöhnlichen BDDs können diese Ähnlichkeiten aufgrund der Verwendung unterschiedlicher Variablen zur Kodierung des lokalen Zustands verschiedener Komponenten nicht ausgenutzt werden. Sehr große Speicherreduktionen lassen sich bei Verwendung von TLEBDDs häufig bei Systemen aus mehreren replizierten Komponenten eines Komponententyps erzielen. Werden sich entsprechende Bits zur Kodierung der lokalen Zustände verschiedener Komponenten in den BDDs der TLEBDDs auf die gleichen reduzierten Variablen abgebildet,

können sich durch die Gleichartigkeit der Komponenten sehr große isomorphe Teilgraphen ergeben. Diese müssen auch bei vielen in einem System vorhandenen replizierten Komponenten vom BDD-Programmpaket nur einmal abgespeichert werden.

In Abschnitt 2.6 dieser Arbeit wurde ein Beispiel eines asynchronen nebenläufigen Systems mit Transitionslokalität aus zwei Komponenten vorgestellt. Für dieses wurde dort in Abbildung 2.13 eine komponentenweise partitionierte Transitionsrelation des Systems, die aus jeweils einem BDD mit den Transitionen jeder Komponente besteht, angegeben. Durch die Kodierung des lokalen Zustands der Komponenten über unterschiedlichen Booleschen Variablen sind keine größeren zwischen den beiden BDDs isomorphen Teilgraphen vorhanden. Für eine Repräsentation der Transitionen der Komponenten durch TLEBDDs, sind in Abbildung 4.1 die BDDs von TLEBDDs mit den Transitionen jeder Komponente angegeben. Die TLEBDDs wurden dort für eine Zuordnung der korrespondierenden tatsächlichen Booleschen Variablen zur Kodierung des lokalen Zustands der Komponenten auf die gleichen reduzierten Variablen der BDDs der TLEBDDs angegeben. Wie auch bei den in der vorliegenden Arbeit angegebenen Definitionen des relationalen Produkts, werden hier aus Veranschaulichungsgründen zur Kodierung von tatsächlichen Variablen für aktuelle Zustände Variablen der Menge $X = \{x_1, x_2, ..., x_N\}$ und für tatsächliche Variablen für Nachfolgerzustände Variablen der Menge $X' = \{x'_1, x'_2, ..., x'_N\}$ verwendet. Die Repräsentation durch tatsächliche Variablen der Menge $X = \{x_1, x_2, ..., x_{2 \cdot N}\}$ (wie bei der Einführung von TLEBDDs in Definition 4.3) kann daraus durch Indextransformation und Vereinigung von Variablen x_i mit den neuen Indizes erhalten werden. In Abschnitt 2.6 wurden aus Gründen der besseren Verständlichkeit für die Booleschen Variablen zur Kodierung des globalen Zustands die Variablen x_g (Variable für aktuellen Zustand) und x'_g (korrespondierende Variable für Nachfolgerzustände) verwendet. Diese können ebenfalls durch entsprechende Transformation in Variablen der Menge $X = \{x_1, x_2, ..., x_{2 \cdot N}\}$ überführt werden. Das gleiche gilt für die in Abbildung 4.1 angegebenen reduzierten Variablen und die Transformation dieser in Variablen der Menge $Y = \{y_1, y_2, ..., y_{2 \cdot (n_g + n_{l_i})}\}$. Bei Verwendung der gerade beschriebenen Variablenmengen ergeben sich für die TLEBDDs aus Abbildung 4.1 folgende Zuordnungslisten zur Zuordnung von reduzierten Variablen auf die diesen entsprechenden tatsächlichen Variablen:

- Zuordnungsliste Komponente 1: [x_{g},x'_{g},x_1,x'_1]
- Zuordnungsliste Komponente 2: [x_{g},x'_{g},x_2,x'_2]

In Abbildung 4.1 sind die sich durch die Verwendung von TLEBDDs ergebenden maximalen isomorphen Teilgraphen, die vom in der vorliegenden Arbeit verwendeten BDD-Programmpaket CUDD [181] nur einmal abgespeichert werden müssen, durch Umrandungen gekennzeichnet. Bei Verwendung von BDDs würden sich die isomorphen Teilgraphen aus inneren Knoten der BDDs, auf die beiden mit der Variable x'_2 markierten Knoten beschränken (siehe Abbildung 2.13).

Um aus einem TLEBDD einen gewöhnlichen BDD zu erhalten, müssen die im BDD des TLEBDDs für reduzierte Variablen vorhandenen Knoten in Knoten für die zu den reduzierten Variablen korrespondierenden tatsächlichen Variablen umbenannt werden.

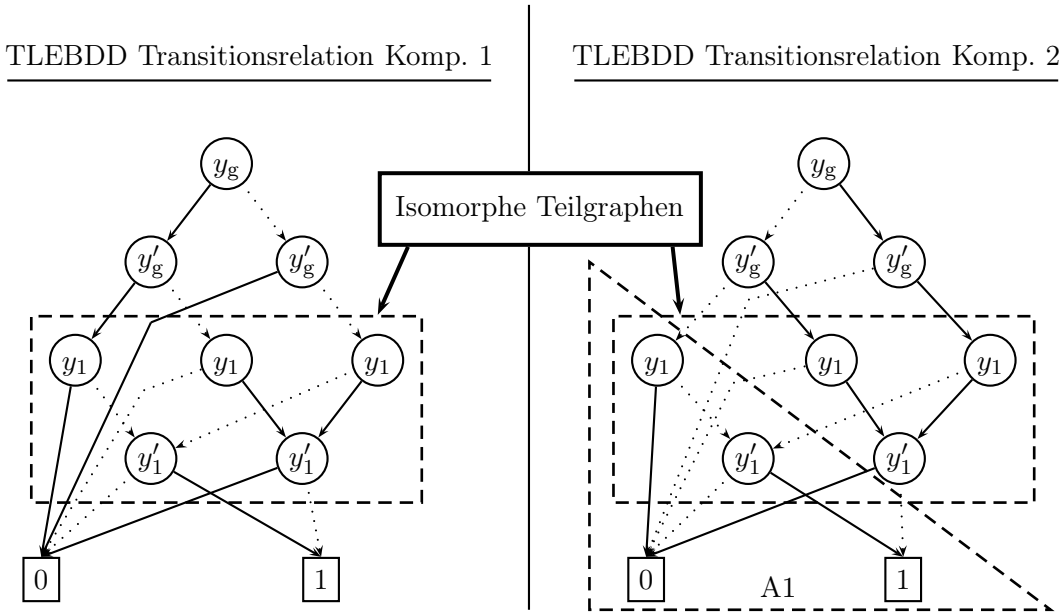

Abbildung 4.1: TLEBDDs zur Repräsentation aller Transitionen von Komponente 1 bzw. von Komponente 2 des Beispielsystems aus Abschnitt 2.6. Als Variablenordnung für die reduzierten Variablen wird hier die Variablenordnung $<_{level_4}$ mit $y_g <_{level_4} y'_g <_{level_4} y_1 <_{level_4} y'_1$ verwendet.

Zusätzlich müssen die fehlenden Muster für Identitätstransformationen eingefügt werden. Für einen TLEBDD für eine Komponente i mit einem BDD über den im Vektor $\vec{y} = (y_1, y_2, ..., y_{2\cdot(n_g+n_{l_i})})$ enthaltenen reduzierten Variablen, kann das Umbenennen in tatsächliche Variablen gemäß der in der Zuordnungsliste angegebenen Zuordnungsinformation erfolgen. Sei $\vec{x}_{\pi_i} = (x_{\pi_i(1)}, x_{\pi_i(2)}, ..., x_{\pi_i(2\cdot(n_g+n_{l_i}))})$ ein mit der Zuordnungsinformation gewonnener Vektor aus Booleschen Variablen, dann kann das Umbenennen mit der in Abschnitt 2.5.1 eingeführten Umbenennungsoperation gemäß $\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}$ durchgeführt werden. Das Einfügen der fehlenden Identitätstransformationen für zusammengehörige Paare von Variablen für aktuelle Zustände und Nachfolgerzustände kann durch Hinzufügen der entsprechenden BDD-Knoten erfolgen. Für verschachtelte Variablenordnungen wäre dafür das Einfügen der in Abschnitt 2.5.3 vorgestellten Identitätsmuster notwendig. Zur Definition der Wirkung des Einfügens von Identitätstransformationen wird in Definition 4.4 die zweistellige Operation Δ eingeführt.

Definition 4.4. (Operation Δ zum Einfügen von Identitätstransformationen) *Die Operation Δ ist eine zweistellige Operation $\Delta(IdVar, H)$, die als Übergabeparameter eine Menge von Variablenpaaren $IdVar$ aus korrespondierenden Variablen für aktuelle Zustände und Nachfolgerzustände und einen BDD H, der keine wesentlichen Variablen die in einem Variablenpaar aus $IdVar$ enthalten sind besitzt, hat. Diese Operation erweitert*

den Definitionsbereich des BDDs H falls nötig um alle in $IdVar$ vorkommenden Variablen und fügt Knoten für Identitätstransformationen in den resultierenden BDD ein.

Sei $H_{exp} = \Delta(IdVar, H)$ das Ergebnis der Anwendung der Operation Δ mit einer Menge von Variablenpaaren $IdVar$ und einem BDD H als Übergabeparametern, $\vec{x}$ ein Vektor mit den Booleschen Eingabevariablen des BDDs H_{exp} und $\vec{x}_{red}$ ein Vektor mit der Teilmenge der im Vektor $\vec{x}$ enthaltenen Booleschen Variablen, die die Eingabevariablen des BDDs H sind. Dann gilt für die durch den BDD $H_{exp} = \Delta(IdVar, H)$ repräsentierte charakteristische Funktion:

$$H_{exp}(\vec{x}) = \begin{cases} 1 & H(\vec{x}_{red}) = 1 \text{ und die beiden Variablen jedes Variablenpaares aus einer} \\ & \text{Variablen für aktuelle Zustände } x_a \text{ und einer Variablen für Nachfolger-} \\ & \text{zustände } x_b \text{ mit } (x_a, x_b) \in IdVar \text{ besitzen die gleichen Werte} \\ 0 & \text{sonst} \end{cases}$$

Ein gewöhnlicher BDD zu einem TLEBDD H für eine Komponente i, kann bei Benutzung der Operation Δ und vorheriger Umbenennung der reduzierten Variablen des TLEBDDs in entsprechende tatsächliche Variablen, durch $\Delta(IdVar, H\{\vec{y} \leftarrow \vec{x}_{\pi_i}\})$ konstruiert werden. Dabei ist IdVar hier als die Menge aller korrespondierenden Paare von Variablen für Identitätstransformationen, für Änderungen des lokalen Zustands der anderen neben Komponente i im System vorhandenen Komponenten, zu wählen. Ein konkreter Algorithmus für die Anwendung der Operation Δ auf einen übergebenen BDD, könnte durch Traversieren des übergebenen BDDs und dem Einfügen der Identitätstransformationen während der Traversierung implementiert werden. Für einen TLEBDD kann der Wahrheitswert zu einer Eingabebelegung der entsprechenden tatsächlichen Variablen, ähnlich wie bei einem gewöhnlichen BDD, durch Durchlaufen des BDDs des TLEBDDs vom Wurzelknoten zu einem Terminalknoten ermittelt werden. Dabei müssen zur Zuordnung der tatsächlichen Variablen auf die diesen entsprechenden reduzierten Variablen noch die Informationen der Zuordnungsliste, sowie die fehlenden Knoten für Identitätstransformationen beachtet werden. Durch die Verwendung von geordneten Zuordnungslisten, die im Folgenden in Definition 4.5 eingeführt werden und die gemäß einer totalen Ordnung über den tatsächlichen Variablen X geordnet sind, kann die Kanonizität von TLEBDDs bei der Repräsentation von Transitionen einer Komponente sichergestellt werden. Bei der Repräsentation von Transitionen verschiedener Komponenten durch TLEBDDs, unterscheiden sich die in den Zuordnungslisten der TLEBDDs enthaltenen tatsächlichen Variablen um die verschiedenen tatsächlichen Variablen zur Kodierung des lokalen Zustands der Komponenten.

Definition 4.5. (Geordnete Zuordnungsliste) *Gegeben sei eine totale Ordnung $\prec$ über den tatsächlichen Variablen X. Dann ist eine n-stellige Zuordnungsliste für eine Komponente i geordnet, wenn $x_{\pi_i(j)} \prec x_{\pi_i(j+1)}$ für alle $1 \leq j < n$ gilt.*

Satz 4.6. (Kanonizität von TLEBDDs) *Seien (G, b_g) und (H, b_h) zwei TLEBDDs mit geordneten Zuordnungslisten, die gemäß einer totalen Ordnung $\prec$ über den tatsächlichen Variablen X geordnet sind, $<_{level}$ die Variablenordnung der BDDs G und H der TLEBDDs und seien g, h Boolesche Funktionen zur Repräsentation von Transitionen einer Komponente K_i eines asynchronen nebenläufigen Systems mit Transitionslokalität. Wenn g durch (G, b_g) und h durch (H, b_h) repräsentiert wird, gilt $g = h$ genau dann, wenn $G = H$ und $b_g = b_h$ gilt.*

Beweis. Seien $G = H$ und $b_g = b_h$. Dann sind die Zuordnungslisten identisch und es gilt $b_g = b_h = [x_{\pi_i(1)}, ..., x_{\pi_i(n)}]$. Bei Expansion der TLEBDDs in gewöhnliche BDDs ergibt sich $g_{exp} = \Delta(IdVar, G\{\vec{y} \leftarrow \vec{x}_{\pi_i}\})$ und $h_{exp} = \Delta(IdVar, H\{\vec{y} \leftarrow \vec{x}_{\pi_i}\})$, wobei Δ wie in Definition 4.4 angegeben definiert ist. $IdVar$ ist die Menge der Variablenpaare für die implizit Identitätstransformationen zu berücksichtigen sind. Dies sind alle tatsächlichen Variablen außer den tatsächlichen Variablen die in der Zuordnungsliste vorkommen, daher ist die Menge $IdVar$ bei der Erzeugung von g_{exp} und h_{exp} gleich. Weil auch $G = H$ gilt, gilt $g_{exp} = h_{exp}$ und deshalb auch $g = h$.
Nun sei $g = h$. Da die Zuordnungslisten der TLEBDDs geordnet sind und für Komponente K_i die gleichen tatsächlichen Variablen den reduzierten Variablen zugeordnet werden müssen, gibt es nur eine eindeutige geordnete Zuordnungsliste. Deshalb gilt $b_g = b_h$. Wäre bei gleicher Variablenordnung $<_{level}$ $G \neq H$, dann müssten die TLEBDDs (G, b_g) bzw. (H, b_h) verschiedene Zuordnungslisten haben, damit $g = h$ gilt. Daher gilt auch $G = H$. □

Damit TLEBDDs und gewöhnliche BDDs bei der Modellprüfung zusammen benutzt werden können, ist es notwendig, dass deren Kombination durch Boolesche Operationen möglich ist. Ein einfacher Ansatz wäre die Expansion eines TLEBDDs zu einem korrespondierenden BDD vor der Ausführung einer Booleschen Operation. Dadurch könnten anschließend die gewöhnlichen Algorithmen zur Kombination von BDDs verwendet werden. Der Aufbau des expandierten BDDs kann aber einiges an Laufzeit in Anspruch nehmen und zusätzlich kann das Speichern dieses BDDs viel Speicherbedarf erfordern. Daher wird in Abschnitt 4.3 der vorliegenden Arbeit für verschachtelte Variablenordnungen ein Algorithmus zur kombinierten Berechnung des relationalen Produkts zwischen einem TLEBDD und einem BDD vorgestellt, bei dessen Verwendung keine vorherige Expansion des TLEBDDs in einen entsprechenden BDD notwendig ist. Die dort verwendete Vorgehensweise kann auch zur Implementierung von Booleschen Operationen zwischen TLEBDDs und BDDs verwendet werden. Vor der Kombination von TLEBDDs und BDDs müssen die Variablenordnung der reduzierten Variablen des TLEBDDs und die der tatsächlichen Variablen allerdings angeglichen werden. Daher wird im nachfolgenden Abschnitt auf verträgliche TLEBDD- und BDD-Variablenordnungen eingegangen.

4.2 Variablenordnungen von TLEBDDs und BDDs

Bei der gemeinsamen Verwendung von TLEBDDs und BDDs bei der Modellprüfung, können prinzipiell verschiedene Variablenordnungen für die reduzierten Variablen des TLEBDDs und die diesen entsprechenden in BDDs verwendeten tatsächlichen Variablen verwendet werden. Allerdings ist es notwendig diese Variablenordnungen vor Operationen mit TLEBDDs und BDDs, wie zum Beispiel bei der Benutzung des Algorithmus zur kombinierten Berechnung des relationalen Produkts zwischen einem TLEBDD und einem BDD, der in Abschnitt 4.3 vorgestellt wird, anzugleichen. Wird im Folgenden von der Variablenordnung eines TLEBDDs gesprochen, so ist damit die für die reduzierten Variablen des BDDs des TLEBDDs verwendete Variablenordnung gemeint. Im Definition 4.7 wird die Konformität einer TLEBDD- und einer BDD-Variablenordnung eingeführt.

Definition 4.7. (**Konformität einer TLEBDD- und einer BDD-Variablenordnung**) *Gegeben sei ein TLEBDD mit Transitionen einer Komponente i, dessen BDD über einer Menge $Y = \{y_1, y_2, ..., y_{2\cdot(n_g+n_{l_i})}\}$ von reduzierten Booleschen Variablen definiert ist. Außerdem seien ein BDD über einer Menge $X = \{x_1, x_2, ..., x_{2\cdot N}\}$ von tatsächlichen Booleschen Variablen, eine Variablenordnung für die Variablen des TLEBDDs $<_{TLEBDD}$ und eine Variablenordnung für die Variablen des BDDs $<_{BDD}$ gegeben. Dann heißen die Variablenordnungen des TLEBDDs $<_{TLEBDD}$ und des BDDs $<_{BDD}$ konform, wenn gilt:*

$$\forall y_i, y_j \in Y \; mit \; y_i \neq y_j : y_i <_{TLEBDD} y_j \Leftrightarrow x_{\pi_i(i)} <_{BDD} x_{\pi_i(j)} \tag{4.2}$$

In Definition 4.7 ist $\pi_i(i)$ die in Definition 4.1 in Abschnitt 4.1 eingeführte Zuordnungsfunktion einer Komponente i. Bei konformen TLEBDD- und BDD-Variablenordnungen kommen die reduzierten Variablen eines TLEBDDs und die diesen entsprechenden tatsächlichen Variablen in beiden Variablenordnungen in der gleichen Reihenfolge vor.
In einem BDD bzw. einem TLEBDD ist die Variablenordnung auf allen Pfaden vom Wurzelknoten zu einem Terminalknoten gleich. Dadurch kann die Verknüpfung von TLEBDDs und BDDs bei Verwendung konformer Variablenordnungen durch gemeinsames Durchlaufen des TLEBDDs und des BDDs beginnend mit ihren Wurzelknoten berechnet werden. Knoten für reduzierte Variablen des TLEBDDs und diesen entsprechende Knoten für tatsächliche Variablen des BDDs, die zur Berechnung des Ergebnisses des relationalen Produkts zusammen betrachtet werden müssen, kommen gemäß der TLEBDD- und BDD-Variablenordnung in der gleichen Reihenfolge vor und können entsprechend verknüpft werden. Soll das relationale Produkt zwischen TLEBDDs und BDDs für mehrere Komponenten berechnet werden, ist es sinnvoll Variablenordnungen zu verwenden, die für alle Komponenten mit denen Bildberechnungen ausgeführt werden konform sind. Dadurch können Umordnungen der Variablenordnungen und der für diese notwendige Zeitaufwand vor Bildberechnungen mit verschiedenen Komponenten eingespart werden. Im nachfolgenden Abschnitt wird ein Algorithmus zur kombinierten Berechnung des relationalen Produkts zwischen einem TLEBDD und einem BDD eingeführt. Dieser kann mit verschachtelten Variablenordnungen, die zusätzlich konform sind, verwendet werden.

4.3 Algorithmus zur Berechnung des relationalen Produkts mit TLEBDDs

In diesem Abschnitt wird für verschachtelte Variablenordnungen ein Algorithmus zur Berechnung des relationalen Produkts für die Nachfolgerberechnung zwischen einem TLEBDD zur Repräsentation einer Menge von Transitionen und einem BDD zur Repräsentation einer Zustandsmenge vorgestellt. Bei Verwendung eines TLEBDDs mit einem BDD $R(\vec{y})$ über einer Menge von reduzierten Variablen zur Repräsentation von Transitionen einer Komponente i, kann das relationale Produkt für die Nachfolgerberechnung durch $\exists\vec{x}[\Delta(IdVar, R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}) \wedge S(\vec{x})]\{\vec{x}' \leftarrow \vec{x}\}$ beschrieben werden. In diesem relationalen Produkt ist $\vec{x}_{\pi_i} = (x_{\pi_i(1)}, x_{\pi_i(2)}, ..., x_{\pi_i(2\cdot(n_g+n_{l_i}))})$ ein aus der Zuordnungsliste der Komponente gewonnener Vektor aus tatsächlichen Variablen, der die Zuordnungen von reduzierten Variablen auf die diesen entsprechenden tatsächlichen Variablen beinhaltet. Durch Ausführen der Umbenennung der reduzierten Variablen des BDDs des TLEBDDs in die diesen entsprechenden tatsächlichen Variablen $\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}$ und der anschließenden Anwendung der in Abschnitt 4.1 eingeführten Operation Δ, die als ersten Übergabeparameter eine Menge $IdVar$ von Variablenpaaren aus korrespondierenden Variablen für aktuelle Zustände und Nachfolgerzustände, für die Identitätsmuster zu berücksichtigen sind, besitzt, kann aus dem in dieser Beschreibung des relationalen Produkts vorhandenen BDD des TLEBDDs ein entsprechender gewöhnlicher BDD erzeugt werden. Anschließend könnte die Ermittlung der Nachfolgerzustände durch den in Abschnitt 3.1 der vorliegenden Arbeit vorgestellten Algorithmus zur kombinierten Berechnung des relationalen Produkts mit BDDs erfolgen. Zu beachten ist noch, dass $\vec{x}_{\pi_i}$ aus Variablen $x_{\pi_i(j)}$ besteht, wohingegen die Vektoren bei der Umbenennungsoperation $\{\vec{x}' \leftarrow \vec{x}\}$ Variablen x'_i bzw. x_i beinhalten. Ebenso sind die Variablen aus $\vec{x}$ die Eingabevariablen des BDD $S(\vec{x})$. Daher liefert die Umbenennungsoperation $\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}$ hier direkt die den Variablen aus $x_{\pi_i(j)}$ entsprechenden Variablen aus $\vec{x}'$ und $\vec{x}$ als Ergebnis. Siehe dazu auch die Beschreibung der Beziehung zwischen diesen Variablenmengen in Abschnitt 4.1.

Im Folgenden wird ein Algorithmus vorgestellt, der die im relationalen Produkt enthaltenen Operationen auf einmal durch direkte Kombination eines TLEBDDs und eines BDDs berechnet. Damit können der Bau des dem TLEBDD entsprechenden möglicherweise großen gewöhnlichen BDDs und die dabei anfallende Laufzeit vermieden werden. Verwendet werden kann der Algorithmus mit verschachtelten Variablenordnungen, bei denen die TLEBDD- und die BDD-Variablenordnung konform sind (siehe Abschnitt 4.2). Auch ist die Benutzung des Algorithmus bei Verwendung von Verfahren zum dynamischen Verändern der Variablenordnung möglich, wenn vor Ausführung des Algorithmus immer verschachtelte und konforme Variablenordnungen erzeugt werden. Die Benutzung des Algorithmus zusammen mit Verfahren zum dynamischen Verändern der Variablenordnung in Verifikationsexperimenten und die dabei für Variablenordnungen verwendeten Einschränkungen werden in Abschnitt 8.3 beschrieben. Pseudocode des Algorithmus zur kombinierten Berechnung des relationalen Produkts mit einem TLEBDD ist in Algorithmus 12 angegeben.

Algorithmus 12: Kombinierte Berechnung des relationalen Produkts $\exists\vec{x}[\Delta(IdVar, R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}) \wedge S(\vec{x})]\{\vec{x}' \leftarrow \vec{x}\}$ mit einem TLEBDD zur Repräsentation von Transitionen einer Komponente i.

```
1  RelProductTLEBDD(TLEBDD_BDD R, MappingList mapping, ROBDD S,
2                   Index compIndex)
3  if IsTerminalCase(R, S, mapping) then
4  |  return TerminalCase(R, S, mapping);
5  else
6  |  if ComputedTableHasEntry(RelProdTLE, R, S, compIndex) then
7  |  |  return ComputedTable(RelProdTLE, R, S, compIndex);
8  |  else
9  |  |  if !IsTerminalVertex(R) then
10 |  |  |  mappedVariable = mapping[VariableRoot(R)];
11 |  |  else
12 |  |  |  mappedVariable = TerminalValue;
13 |  |  υ = TopVariable(mappedVariable, VariableRoot(S));
14 |  |  if υ ==mappedVariable then
15 |  |  |  w = VariableRoot(R);
16 |  |  |  R_{w0} = R|_{w=0};
17 |  |  |  R_{w1} = R|_{w=1};
18 |  |  else
19 |  |  |  R_{w0} = R_{w1} = R;
20 |  |  R_1 = RelProductTLEBDD(R_{w1}, mapping, S|_{υ=1}, compIndex);
21 |  |  R_0 = RelProductTLEBDD(R_{w0}, mapping, S|_{υ=0}, compIndex);
22 |  |  if R_0 == R_1 then
23 |  |  |  return R_1;
24 |  |  else
25 |  |  |  if IsIdentityPatternVariable(υ, compIndex) then
26 |  |  |  |  Result = FindOrAddUniqueTable(υ, R_0, R_1);
27 |  |  |  else
28 |  |  |  |  if IsCurrentStateVar(υ) then
29 |  |  |  |  |  Result = APPLY(Or, R_0, R_1);
30 |  |  |  |  else
31 |  |  |  |  |  Result = FindOrAddUniqueTable(GetCurrentStateVar(υ), R_0,
32 |  |  |  |  |                                R_1);
33 |  |  InsertComputedTable(RelProdTLE, R, S, compIndex, Result);
34 |  |  return Result;
```

Der hier vorgestellte Algorithmus führt die Berechnung des relationalen Produkts nach dem in Abschnitt 2.2.4 für BDDs angegebenen rekursiven Berechnungsschema durch. Bei diesem werden rekursiv Aufspaltungen in einfachere Teilprobleme durchgeführt, die aus Kofaktoren für eine Belegung der Top-Variablen eines Rekursionsschritts mit den Booleschen Werten bestehen. Die für Teilprobleme ermittelten Teilergebnisse werden zum Ergebnis eines Rekursionsschritts kombiniert. Vor rekursiven Aufrufen des Algorithmus für Teilprobleme, wird wie beim in Abschnitt 2.2.4 in Algorithmus 2 angegebenen APPLY-Algorithmus in einem Rekursionsschritt wieder geprüft, ob ein Terminalfall der rekursiven Berechnung vorliegt. Bei nicht vorhandenem Terminalfall wird abgefragt, ob das Ergebnis der in einem Rekursionsschritt auszuführenden Berechnung bereits ermittelt wurde und noch in der Ergebnistabelle gespeichert ist. Falls kein Terminalfall vorliegt und das zu berechnende Ergebnis in der Ergebnistabelle nicht gefunden wurde, werden weitere rekursive Aufrufe des Algorithmus durchgeführt. Dazu wird im hier vorgestellten Algorithmus zuerst die der reduzierten Variablen des Wurzelknotens des BDDs des TLEBDDs entsprechende tatsächliche Variable *mappedVariable* ermittelt. Für diese tatsächliche Variable wird der Wurzelknoten des TLEBDDs im aktuellen Rekursionsschritt berücksichtigt. Die Korrektheit der von Algorithmus 12 bei der rekursiven Berechnung verwendeten Aufspaltung in Teilprobleme und der Kombination von Teilergebnissen zum Ergebnis eines Rekursionsschritts wird in Abschnitt 4.4 gezeigt. Beim rekursiven Berechnungsschema des Algorithmus werden in Abhängigkeit von der Top-Variablen υ unterschiedliche Vorgehensweisen verwendet. Für einen Rekursionsschritt mit einer Top-Variable, die eine Variable zur Kodierung des lokalen Zustands einer anderen Komponente als der für die der betrachtete TLEBDD gebaut wurde ist, wird die Korrektheit der im Algorithmus verwendeten Aufspaltung im Beweis zu Satz 4.8 in Abschnitt 4.4 gezeigt. Für die Komponente, für die der betrachtete TLEBDD gebaut wurde, ist eine solche Top-Variable eine Variable zur Kodierung einer Identitätstransformation. Die Aufspaltung in Teilprobleme erfolgt im Algorithmus für Top-Variablen für diese Identitätstransformationen immer gemeinsam für ein Variablenpaar aus Variable für aktuelle Zustände und korrespondierender Variable für Nachfolgerzustände. In der nachfolgend für diese Aufspaltung angegebenen Gleichung aus Satz 4.8 ist $IdVar'$ die Menge $IdVar$ ohne das Variablenpaar aus Variable für aktuelle Zustände υ und korrespondierender Variable für Nachfolgerzustände υ', bezüglich dem die Aufspaltung in Teilprobleme angegeben ist ($IdVar' = IdVar - \{(\upsilon,\ \upsilon')\}$):

$$\begin{aligned}
&\exists\vec{x}[\Delta(IdVar, R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}) \wedge S(\vec{x})]\{\vec{x}' \leftarrow \vec{x}\} \\
&= \upsilon \cdot ((\exists\vec{x}(\Delta(IdVar', R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}) \wedge S(\vec{x})|_{\upsilon=1}))\{\vec{x}' \leftarrow \vec{x}\}) \vee \\
&\overline{\upsilon} \cdot ((\exists\vec{x}(\Delta(IdVar', R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}) \wedge S(\vec{x})|_{\upsilon=0}))\{\vec{x}' \leftarrow \vec{x}\})
\end{aligned}$$

Im BDD des TLEBDDs sind keine Knoten für Identitätstransformationen zur Kodierung von Änderungen des lokalen Zustands von Komponenten, für die ein betrachteter TLEBDD nicht gebaut wurde, vorhanden. Daher ist die Top-Variable in einem solchen Rekursionsschritt immer eine Variable für aktuelle Zustände eines Wurzelknotens des BDDs $S(\vec{x})$ und der BDD des TLEBDDs $R(\vec{y})$ wird bei den rekursiven Aufrufen unverändert als Übergabeparameter verwendet. Andere Vorgehensweisen werden im Algorithmus bei der Aufspaltung in Teilprobleme in Rekursionsschritten verwendet, bei denen die Top-

Variable des Rekursionsschritts eine Variable zur Kodierung des globalen oder lokalen Zustands der Komponente, für die der betrachtete TLEBDD gebaut wurde, ist. Hier werden in Algorithmus 12 in Abhängigkeit davon, ob die Top-Variable eine Variable für aktuelle Zustände ($\upsilon \in X$) oder eine Variable für Nachfolgerzustände ($\upsilon \in X'$) ist, unterschiedliche Aufspaltungen in Teilprobleme verwendet. Diese sind im Folgenden aufgeführt und deren Korrektheit wird in Abschnitt 4.4 im Beweis zu Satz 4.9 gezeigt:

$$\exists \vec{x}[\Delta(IdVar, R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}) \wedge S(\vec{x})]\{\vec{x}' \leftarrow \vec{x}\} =$$

$$\begin{cases} (\exists \vec{x}(\Delta(IdVar, R(\vec{y})|_{y_{\pi_i^{-1}(\upsilon)}=1}\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}) \wedge S(\vec{x})|_{\upsilon=1}))\{\vec{x}' \leftarrow \vec{x}\} \vee \\ (\exists \vec{x}(\Delta(IdVar, R(\vec{y})|_{y_{\pi_i^{-1}(\upsilon)}=0}\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}) \wedge S(\vec{x})|_{\upsilon=0}))\{\vec{x}' \leftarrow \vec{x}\} & \upsilon \in X \\ \\ (\upsilon)\{\vec{x}' \leftarrow \vec{x}\} \cdot (\exists \vec{x}(\Delta(IdVar, R(\vec{y})|_{y_{\pi_i^{-1}(\upsilon)}=1}\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}) \wedge S(\vec{x}))) & \\ \{\vec{x}' \leftarrow \vec{x}\} \vee (\overline{\upsilon})\{\vec{x}' \leftarrow \vec{x}\} \cdot (\exists \vec{x}(\Delta(IdVar, R(\vec{y})|_{y_{\pi_i^{-1}(\upsilon)}=0}\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}) & \upsilon \in X' \\ \wedge S(\vec{x})))\{\vec{x}' \leftarrow \vec{x}\} & \end{cases}$$

Im Folgenden wird der in Algorithmus 12 angegebene Algorithmus *RelProductTLEBDD()* detaillierter beschrieben. Der Algorithmus besitzt als Übergabeparameter den BDD R eines TLEBDDs, mit dem eine Menge von Transitionen einer Komponente repräsentiert wird und die Zuordnungsliste *mapping* des TLEBDDs. Außerdem bekommt er einen BDD S mit der zu explorierenden Zustandsmenge und den Index *compIndex* der Komponente, für die im TLEBDD Transitionen gespeichert sind, übergeben. Im Algorithmus wird zuerst geprüft, ob bei der in einem Rekursionsschritt durchzuführenden Berechnung ein Terminalfall der rekursiven Berechnung vorliegt (Zeile 3). Liegt ein Terminalfall vor, wird das Ergebnis des Terminalfalls durch Aufruf der Funktion *TerminalCase()* ermittelt und als Rückgabewert des Rekursionsschritts zurückgegeben (siehe Zeile 4). Die Funktion *TerminalCase()* ist in Algorithmus 13 dargestellt. Sie beinhaltet die verwendeten Terminalfälle, auf deren Vorliegen auch die Funktion *IsTerminalCase()* prüft, und die Rückgabewerte der Terminalfälle. Wie auch bei der kombinierten Berechnung des relationalen Produkts zwischen BDDs liegen Terminalfälle vor, wenn der Wurzelknoten des BDDs R oder der des BDDs S ein Terminalknoten für den Booleschen Wert 0 ist. Die im relationalen Produkt enthaltene UND-Verknüpfung und damit auch die noch durchzuführende Teilberechnung des relationalen Produkts werten dann, wie in Abschnitt 3.1 für das relationale Produkt zwischen BDDs beschrieben, unabhängig vom Wert eines im anderen BDD möglicherweise später erreichten Terminalknotens auf den Wert 0 aus. Daher ändern auch im BDD des TLEBDDs fehlende und möglicherweise zu berücksichtigende Identitätsmuster den Rückgabewert dieses Terminalfalls nicht und Algorithmus 13 gibt für diesen Terminalfall den Terminalknoten für den Booleschen Wert 0 zurück (siehe Zeilen 2 und 3 in Algorithmus 13).

Weitere Terminalfälle liegen vor, wenn der Wurzelknotens des BDDs R des TLEBDDs ein Terminalknoten für den Booleschen Wert 1 ist (Zeile 4 in Algorithmus 13). In einem TLEBDD findet keine explizite Repräsentation von Identitätstransformationen für die lokalen Zustände anderer Komponenten als der Komponente, für die der TLEBDD

Algorithmus 13: Funktion *TerminalCase()*, die bei einem in Algorithmus 12 aufgetretenen Terminalfall den BDD mit dem Ergebnis des Terminalfalls berechnet und zurückgibt.

```
1 TerminalCase(TLEBDD_BDD R, ROBDD S, MappingList mapping)
2 if IsTerminalVertexZero(R) || IsTerminalVertexZero(S) then
3     return BDDTerminalVertexZero;
4 if IsTerminalVertexOne(R) then
5     if OnlyGlobalOrLocalVariablesFollow(VariableRoot(S), mapping) ||
6       IsTerminalVertexOne(S) then
7         return BDDTerminalVertexOne;
8     if OnlyIdentityVariablesFollow(VariableRoot(S), mapping) then
9         return S;
```

gebaut wurde, statt. Es wird für zusammengehörige Paare von tatsächlichen Boolesche Variablen zur Kodierung solcher Identitätstransformationen implizit die Interpretation als Identitätstransformation angenommen. Im BDD des TLEBDDs über den reduzierten Variablen werden die Reduktionsregeln für BDDs verwendet. Daher ist ein dort auf einem Berechnungspfad für eine reduzierte Variable fehlender Knoten auf die Anwendung einer Reduktionsregel zurückzuführen. Ist nun der Wurzelknoten des BDDs R in einem Rekursionsschritt ein Terminalknoten für den Booleschen Wert 1, dann können in der Variablenordnung noch zu evaluierende tatsächliche Variablen bezüglich des TLEBDDs diese beiden verschiedenen Interpretationen besitzen. Für die unterschiedlichen bei Repräsentation durch einen TLEBDD möglichen Interpretationen, können sich bei einem im BDD S für eine Variable vorhandenen Knoten verschiedene Ergebnisse beim relationalen Produkt ergeben. Wenn jedoch der Wurzelknoten des BDDs S ebenfalls ein Terminalknoten ist, so ist auf einem Berechnungspfad kein Knoten mehr zu betrachten und das Ergebnis ist für beide Interpretationen gleich. Ist der Wurzelknoten des BDDs S der Terminalknoten für den Booleschen Wert 1, kann daher der Terminalknoten für den Booleschen Wert 1 als Ergebnis eines Rekursionsschritts zurückgegeben werden (Zeile 6). Andernfalls muss der BDDs S bis zum Auftreten eines direkt entscheidbaren Terminalfalls weiter durchlaufen werden, um für jeden Rekursionsschritt die zu berücksichtigende Interpretation zu ermitteln. Als weitere Terminalfälle werden daher nur Fälle verwendet, bei denen in der Variablenordnung ab der tatsächlichen Variablen des Wurzelknotens des BDDs S nur Variablen mit der gleichen Interpretation im TLEBDD vorkommen. Dafür kann das Ergebnis eines Rekursionsschritts direkt entschieden werden. Folgen ab der Variablen des Wurzelknotens des BDDs S in der Variablenordnung nur noch Variablen zur Kodierung des globalen Zustands oder des lokalen Zustands der Komponente, für die der TLEBDD wurde, so ist der Rückgabewert dieses Terminalfalls der Terminalknoten mit dem Booleschen Wert 1 (Zeile 5). Im BDD S kommen dann nur Knoten für Variablen für aktuelle Zustände vor. Diese können bei im BDD des TLEBDDs aufgrund der Anwendung der Reduktionsregeln fehlenden Knoten auf beliebige Werte für Variablen für Nachfolgerzu-

stände abgebildet werden. Nach der existentiellen Quantifizierung und der Umbenennung von Knoten für Variablen für Nachfolgerzustände in entsprechende Knoten für Variablen für aktuelle Zustände, resultiert der Terminalknoten mit dem Booleschen Wert 1 als Rückgabewert.

Der BDD S besitzt nur Knoten für Variablen für aktuelle Zustände. Kommen daher nach der Variable des Wurzelknotens des BDDs S nur noch Variablen für Identitätstransformationen in der Variablenordnung vor, sind die für Variablen für Nachfolgerzustände erlaubten Werte die für die korrespondierenden Variablen für aktuelle Zustände erlaubten Werte. Nach der existentiellen Quantifizierung und der Umbenennung von Variablen für Nachfolgerzustände in Variablen für aktuelle Zustände, ergibt sich für einen solchen Terminalfall der BDD S als Rückgabewert (Zeile 8). Die Korrektheit der hier beschriebenen Terminalfälle, in denen der Wurzelknoten des BDDs R ein Terminalknoten mit dem Booleschen Wert 1 ist, wird in Abschnitt 4.5 gezeigt. Zur schnellen Entscheidung, ob ab einer Variablen in der Variablenordnung nur noch Variablen mit einer Interpretation vorkommen, kann dies für eine Variablenordnung und für jede Komponente einmal ermittelt und abgespeichert werden. Eine Aktualisierung der entsprechenden Werte ist nur bei einer Veränderung der Variablenordnung notwendig.

Ist in einem Rekursionsschritt kein Terminalfall vorhanden wird überprüft, ob das im aktuellen Aufruf des Algorithmus zu berechnende Teilergebnis des relationalen Produkts bereits berechnet wurde und noch in der Ergebnistabelle gespeichert ist (Zeile 6). Damit nur korrekte Ergebnisse von der Ergebnistabelle zurückgeliefert werden, werden als Übergabeparameter bei Zugriffen auf diese die Art der durchzuführenden Operation (hier relationales Produkt zur Kombination eines TLEBDDs und eines BDDs (*RelProdTLE*)), der BDD R für die Menge von Transitionen, der BDD S für die Zustandsmenge und der Index der Komponente *compIndex*, für die der TLEBDD gebaut wurde, verwendet. Im Vergleich mit dem in Abschnitt 3.1 vorgestellten Algorithmus zur kombinierten Berechnung des relationalen Produkts zwischen BDDs, wird hier der Komponentenindex *compIndex* als zusätzlicher Übergabeparameter benutzt. Dies ist notwendig, damit es bei Zugriffen auf die Ergebnistabelle mit den gleichen Wurzelknoten des BDDs R und des BDDs S aber für unterschiedliche Komponenten nicht zur Rückgabe falscher Ergebnisse aus der Ergebnistabelle kommen kann. Für unterschiedliche Komponenten kann der Wurzelknoten des BDDs R durch die sich für die Komponenten unterscheidenden Zuordnungslisten auf verschiedene tatsächliche Variablen abgebildet werden. Dadurch müsste der Wurzelknoten für die Komponenten mit Knoten für unterschiedliche tatsächliche Variablen im BDD S kombiniert werden, was zu unterschiedlichen Ergebnissen der Berechnung führen kann. Durch die zusätzliche Berücksichtigung des Komponentenindexes bei Zugriffen auf die Ergebnistabelle kann sichergestellt werden, dass für einen Knoten R bei einem Eintrag in der Ergebnistabelle nur eine Zuordnung auf die tatsächlichen Variablen gültig ist.

Wenn in der Ergebnistabelle kein Eintrag für die aktuell zu berechnende Operation gefunden wurde, wird die rekursive Berechnung des relationalen Produkts durch Aufspaltung in einfachere Teilprobleme und deren Lösung durch rekursive Aufrufe des Algorithmus *RelProductTLEBDD()* fortgesetzt. Dazu wird mit Hilfe der Zuordnungsliste zunächst die zur reduzierten Variablen des Wurzelknotens des BDDs des TLEBDDs korrespondierende

tatsächliche Variable *mappedVariable* ermittelt (Zeile 10 in Algorithmus 12). Falls es sich beim Wurzelknoten des BDDs R des TLEBDDs um einen Terminalknoten handelt, wird der Wert von *mappedVariable* auf den Wert *TerminalValue* gesetzt (Zeile 12 in Algorithmus 12). Dadurch kann der Wurzelknoten des BDDs R bei der Auswahl der tatsächlichen Top-Variablen eines Rekursionsschritts korrekt berücksichtigt werden. Diese wird in Zeile 13 von Algorithmus 12 bestimmt. Die Top-Variable ist eine tatsächliche Variable. Sie wird deshalb aus der der reduzierten Variablen des Wurzelknotens des BDDs R entsprechenden tatsächlichen Variablen *mappedVariable* und der Variablen des Wurzelknotens des BDDs S ausgewählt. Sie ist die in der verwendeten Variablenordnung höher angeordnete Variable der beiden tatsächlichen Variablen. Bezüglich dieser Top-Variablen werden nachfolgend Kofaktoren der BDDs R und S für die weitere Aufspaltung in Teilprobleme ermittelt. Da der BDD R des TLEBDDs nicht über den tatsächlichen Variablen, sondern über einer reduzierten Variablenmenge definiert ist, müssen Kofaktoren dieses BDDs bezüglich der reduzierten Variablen berechnet werden. Dies ist notwendig, wenn die Top-Variable die der reduzierten Variablen des Wurzelknotens des BDDs R entsprechende tatsächliche Variable ist, da der Wurzelknoten des BDDs R dann für die aktuelle tatsächliche Top-Variable zu berücksichtigen ist. Im Algorithmus wird dazu die reduzierte Variable w des Wurzelknotens des BDDs R in Zeile 15 vor der Ermittlung der Kofaktoren R_{w_0} bzw. R_{w_1}, für eine Belegung dieser Variable mit den beiden Booleschen Werten, berechnet. Entspricht die Top-Variable nicht der tatsächlichen Variablen *mappedVariable* zur reduzierten Variablen des Wurzelknotens des BDDs R, kommt die tatsächliche Variable *mappedVariable* in der verwendeten Variablenordnung später als die Top-Variable des aktuellen Rekursionsschritts vor. Der BDD R besitzt dann im aktuellen Rekursionsaufruf keinen Knoten für eine der Top-Variablen entsprechende reduzierte Variable. Daher müssen für diesen Rekursionsschritt keine neuen Knoten für Kofaktoren des BDDs R berechnet werden und der BDD R kann unverändert für die beiden anschließend durchgeführten rekursiven Aufrufe des Algorithmus verwendet werden ($\mathrm{R}_{w_0} = R$ bzw. $\mathrm{R}_{w_1} = R$, siehe Zeile 19).

Die rekursiven Aufrufe des Algorithmus zur Ermittlung der Ergebnisse der sich durch die Aufspaltung in Teilprobleme ergebenden Berechnungen, sind in den Zeilen 20 und 21 des Algorithmus zu finden. Bei diesen werden die sich durch Belegung der Top-Variable des aktuellen Rekursionsschritts mit Booleschen Werten ergebenden Kofaktoren der Teilprobleme als Übergabeparameter verwendet (R_{w_1} und $\mathrm{S}|_{v=1}$ bzw. R_{w_0} und $\mathrm{S}|_{v=0}$). Nach Ermittlung der Ergebnisse der rekursiven Aufrufe findet die Zusammensetzung der berechneten Teilergebnisse R_1 und R_0 zum Ergebnis eines Rekursionsschritts statt. Sowohl die Korrektheit der verwendeten Aufspaltung in Teilprobleme, als auch die der im Algorithmus angewandten Kombination der Teilergebnisse zum Ergebnis eines Rekursionsschritts, werden in Abschnitt 4.4 gezeigt. Eine Beschreibung dieser erfolgt im Folgenden. Sind die BDDs der zuvor ermittelten Teilresultate R_0 und R_1 gleich, gibt der Algorithmus den BDD eines der Resultate als Ergebnis des Rekursionsschritts zurück (Zeilen 22 und 23 in Algorithmus 12). In diesem Fall ist das resultierende Teilergebnis unabhängig vom Wert der Top-Variablen gleich und es muss aufgrund der bei BDDs verwendeten Reduktionsregeln kein Knoten für die Top-Variable des Rekursionsschritts erzeugt werden. Zur Resultats-

ermittlung bei unterschiedlichen Teilergebnissen R_0 und R_1, verwendet der Algorithmus drei verschiedene Vorgehensweisen.

Ist die Top-Variable eine Variable zur Kodierung einer Identitätstransformation für die Änderungen des lokalen Zustands einer anderen Komponente, als der für die der verwendete TLEBDD gebaut wurde, dann ist das Ergebnis eines Rekursionsschritts ein neuer Knoten mit den Teilergebnissen R_0 und R_1 als Söhnen. Das die Variable eine Variable einer solchen Identitätstransformation ist, wird im Algorithmus mit der Funktion *IsIdentityPatternVariable()* berechnet. Die Erzeugung des neuen Knotens geschieht dabei durch Aufruf der Funktion *FindOrAddUniqueTable()* (siehe Zeile 26), deren Implementierung in Algorithmus 3 in Abschnitt 2.2.4 angegeben ist und die dort auch beschrieben wurde. Knoten für Variablen zur Kodierung solcher Identitätstransformationen können nur im BDD S zur Repräsentation der Zustandsmenge vorkommen, da in TLEBDDs keine Identitätstransformationen für diese vorhanden sind. Weil im BDD S nur Knoten für Variablen für aktuelle Zustände vorhanden sind, tritt dieser Fall der Resultatskombination auch nur bei Variablen für aktuelle Zustände als Top-Variablen auf. Aufgrund der Identitätstransformation müssen diese Variablen auch in auf dem aktuellen Berechnungspfad zu berechnenden Nachfolgerzuständen die gleichen Werte besitzen. Daher findet für diese Variablen keine Wertänderung statt und die rekursiv ermittelten Teilergebnisse sind jeweils für die zu ihrer Berechnung verwendeten Belegungen der Top-Variable gültig.

Für eine Top-Variable zur Kodierung des globalen oder lokalen Zustands der Komponente, für die der TLEBDD gebaut wurde, ist die Art der Berechnung des Resultats eines Rekursionsschritts davon abhängig, ob es sich bei der Top-Variable um eine Variable für aktuelle Zustände oder um eine Variable für Nachfolgerzustände handelt. Ist die Variable eine Variable für aktuelle Zustände ($v \in X$), wird im Algorithmus direkt die bezüglich der Variablen für aktuelle Zustände notwendige existentielle Quantifizierung ($\exists \vec{x}$) durchgeführt. Die existentielle Quantifizierung bezüglich einer Booleschen Variablen x_i, kann durch Veroderung der sich durch Belegung dieser Variablen mit den beiden Booleschen Werten ergebenden Kofaktoren berechnet werden ($\exists_{x_i} f = f|_{x_i=1} \vee f|_{x_i=0}$, siehe auch Abschnitt 2.5.1). Im Algorithmus wird zur Veroderung der in Abschnitt 2.2.4 eingeführte Algorithmus *APPLY* verwendet (siehe Zeile 29 in Algorithmus 12). Für eine Variable für Nachfolgerzustände ($v \in X'$) zur Kodierung des globalen Zustands oder des lokalen Zustands der Komponente, für die der TLEBDD gebaut wurde, wird bei der Resultatskombination direkt die im relationalen Produkt notwendige Umbenennung von Knoten für Variablen für Nachfolgerzustände in entsprechende Knoten für Variablen für aktuelle Zustände durchgeführt. Dazu wird zur Erzeugung des Resultatsknotens die schon bei der Berechnung des Resultats für Top-Variablen für Identitätstransformationen benutzte Funktion *FindOrAddUniqueTable()* aus Abschnitt 2.2.4 verwendet. Die Umbenennung wird durch Erzeugung eines Knotens für die zur Top-Variable korrespondierende Variable für aktuelle Zustände, statt der Erzeugung eines Knoten für die Top-Variable für Nachfolgerzustände durchgeführt (Zeile 31). Ermittelt wird die korrespondierende Variable für aktuelle Zustände durch die Funktion *GetCurrentStateVar()*. Diese Variable kann in verschachtelten Variablenordnungen, für die dieser Algorithmus entwickelt wurde, die direkt vor oder nach der Top-Variablen in der Variablenordnung vorkommende benach-

barte Variable sein. Nach der Ermittlung des Resultats eines Rekursionsschritts durch Kombination der berechneten Teilergebnisse, wird das Ergebnis des Rekursionsschritts zur Ermöglichung der Beschleunigung nachfolgender Berechnungen in die Ergebnistabelle eingetragen (Zeile 33) und zurückgegeben.

Nachfolgend wird die Vorgehensweise des Algorithmus mit einem in Abbildung 4.2 dargestellten Beispielablauf veranschaulicht. Der Beispielablauf wurde für das in Abschnitt 2.6 vorgestellte Beispielsystem erstellt. Zum Beispielsystem wurden TLEBDDs zur Repräsentation aller Transitionen der beiden im System vorhandenen Komponenten in Abbildung 4.1 in Abschnitt 4.1 angegeben. Der *1. Aufruf* des Algorithmus wird im Beispielablauf mit dem in Abbildung 4.1 mit *A1* gekennzeichneten umrandeten Ausschnitt des TLEBDDs zur Repräsentation der Transitionsrelation von Komponente 2 und dem in Abbildung 2.14 in Abschnitt 2.6 ebenfalls mit *A1* gekennzeichneten Ausschnitt des BDDs zur Repräsentation eines Anfangszustands als Eingabeparametern durchgeführt. Der Ablauf des Algorithmus mit diesen Eingabe-BDDs wird im Folgenden ohne die Verwendung der Ergebnistabelle beschrieben. Im *1. Aufruf* des Algorithmus liegt kein Terminalfall vor, weshalb die der reduzierten Variablen des Wurzelknotens des BDDs R des TLEBDDs für Komponente 2 entsprechende tatsächliche Variable *mappedVariable* (x_2) und die *Top-Variable* des Rekursionsschritts (x_2) ermittelt werden. Bezüglich dieser Top-Variablen werden anschließend Kofaktoren des BDDs R ($\mathrm{R}|_{y_1=0}$ bzw. $\mathrm{R}|_{y_1=1}$) und des BDDs S ($\mathrm{S}|_{x_2=0}$ bzw. $\mathrm{S}|_{x_2=1}$) ermittelt. Die Kofaktoren des BDDs des TLEBDDs werden dabei bezüglich der der Top-Variablen entsprechenden reduzierten Variablen (y_1) berechnet.

Der *2. Aufruf* des Algorithmus wird mit den im *1. Aufruf* ermittelten Kofaktoren $\mathrm{R}|_{y_1=1}$ und $\mathrm{S}|_{x_2=1}$ als Übergabeparametern durchgeführt. Da beide Übergabeparameter Terminalknoten für den Booleschen Wert 0 sind, liegt hier ein Terminalfall mit dem Terminalknoten für den Booleschen Wert 0 als Rückgabewert vor. Anschließend erfolgt der *3. Aufruf* des Algorithmus mit den im *1. Aufruf* ermittelten Kofaktoren $\mathrm{R}|_{y_1=0}$ und $\mathrm{S}|_{x_2=0}$. Hier liegt kein Terminalfall vor und die Variable x'_2 ist sowohl *Top-Variable* als auch die der reduzierten Variablen des Wurzelknotens des aktuellen BDDs R entsprechende tatsächliche Variable *mappedVariable*. Bei der Kofaktorbildung bezüglich dieser Top-Variablen, bzw. der entsprechenden reduzierten Variablen des TLEBDDs, resultieren für alle vier Kofaktoren Terminalknoten. Diese führen in den in diesem Rekursionsschritt erfolgenden weiteren rekursiven Aufrufen direkt zu Terminalfällen.

Im *4. Aufruf* des Algorithmus tritt ein Terminalfall mit dem Rückgabewert 1 auf, da beide Übergabeparameter Terminalknoten für den Booleschen Wert 1 sind. Dadurch ergibt sich für diesen Rekursionsschritt der Terminalknoten für den Booleschen Wert 1 als Rückgabewert. Beim *5. Aufruf* ist hingegen der Wurzelknoten des BDDs für den Kofaktor $\mathrm{R}|_{y'_1=0}$ ein Terminalknoten für den Booleschen Wert 0, wodurch der Rückgabewert dieses Rekursionsschritts durch Auftreten des entsprechenden Terminalfalls der Terminalknoten für den Wert 0 ist. Die *Top-Variable* des *3. Aufrufs* des Algorithmus (x'_2) ist eine Variable zur Kodierung von Nachfolgerzuständen. Damit wird in diesem Rekursionsschritt bei der Resultatskombination im Algorithmus ein neuer Knoten für die zu dieser *Top-Variable* für Nachfolgerzustände korrespondierende Variable für aktuelle Zustände, mit den vom *4. Aufruf* und vom *5. Aufruf* erhaltenen Rückgabewerten als Söhnen, gebaut

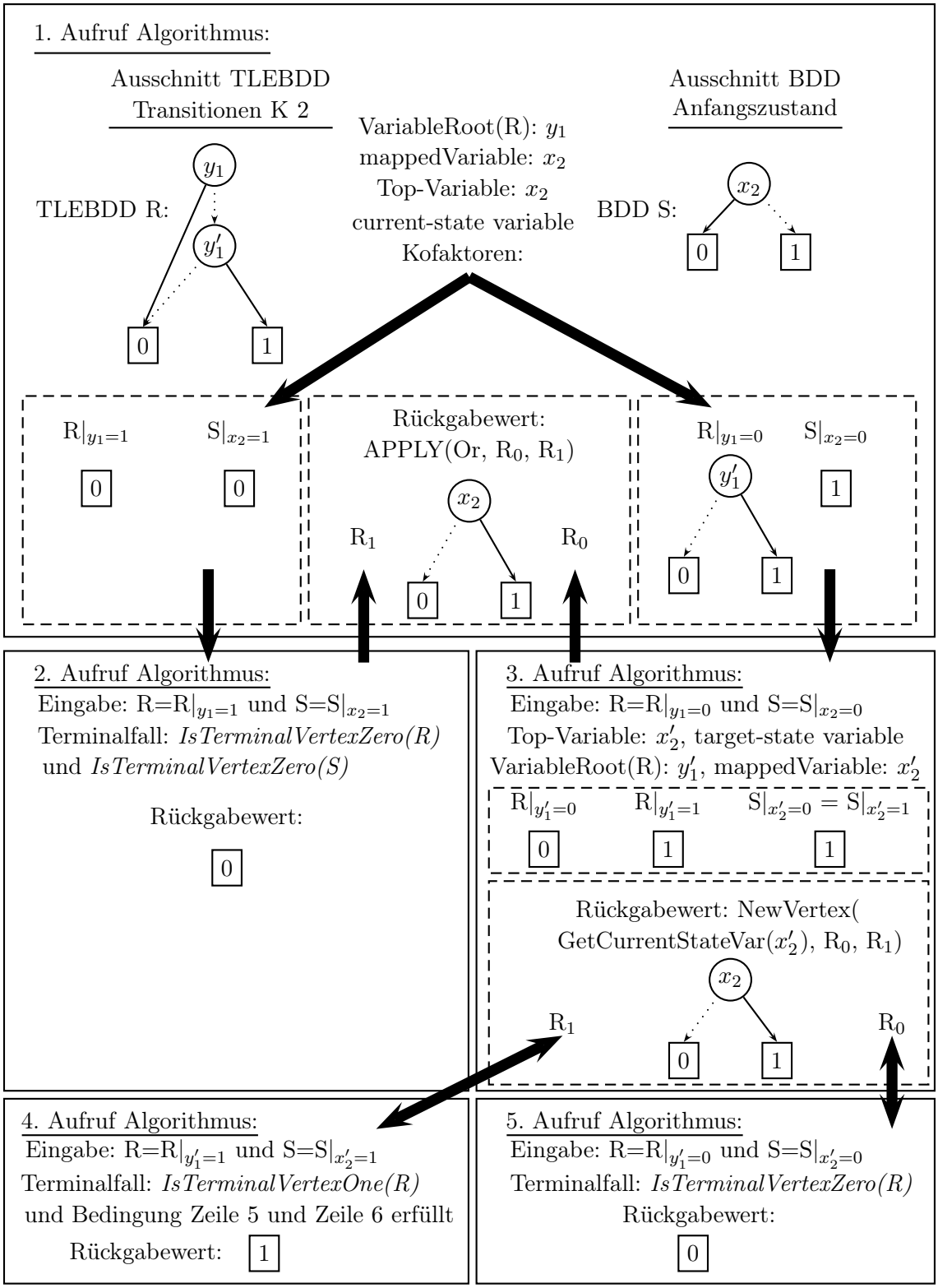

Abbildung 4.2: Beispiel zur Bildberechnung mit dem Algorithmus zur kombinierten Berechnung des relationalen Produkts mit einem TLEBDD.

(FindOrAddUniqueTable(GetCurrentStateVar(x'_2), R_0, R_1), siehe Zeile 31 im Algorithmus) und zurückgegeben. Anschließend wird der Rückgabewert des *1. Aufrufs* des Algorithmus berechnet. Bei der Top-Variable dieses Rekursionsschritts handelt es sich um eine Variable für aktuelle Zustände, daher werden zur Ermittlung des Rückgabewerts des Rekursionsschritts die Rückgabewerte des *2. Aufrufs* und des *3. Aufrufs* durch die ODER-Verknüpfung kombiniert (*APPLY(Or, R_0, R_1)*, siehe Zeile 29 im Algorithmus). Das Resultat ist das Ergebnis der Berechnung des relationalen Produkts, mit den beiden in der Abbildung für den *1. Aufruf* des Algorithmus angegebenen Entscheidungsdiagrammen als Übergabeparametern.

4.4 Korrektheit des Algorithmus

Im vorangehenden Abschnitt wurde ein Algorithmus zur kombinierten Berechnung des relationalen Produkts für die Nachfolgerberechnung $(\exists\vec{x}[\Delta(IdVar, R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}) \wedge S(\vec{x})]\{\vec{x}' \leftarrow \vec{x}\})$ zwischen einem TLEBDD zur Repräsentation einer Menge von Transitionen und einem BDD zur Repräsentation einer zu explorierenden Zustandsmenge präsentiert. Hier wird die Korrektheit der in den Rekursionsschritten des Algorithmus verwendeten Aufspaltung in Teilprobleme und der Kombination der für diese ermittelten Teilergebnisse zum Ergebnis eines Rekursionsschritts gezeigt. Zusammen mit der Korrektheit der im Algorithmus verwendeten Terminalfälle, die in Abschnitt 4.5 erläutert wird, und der in Abschnitt 4.3 beschriebenen Eindeutigkeit von Zugriffen auf die Ergebnistabelle folgt die Korrektheit des vom Algorithmus verwendeten Vorgehens. Da der Algorithmus in Abschnitt 4.3 für verschachtelte Variablenordnungen (siehe Abschnitt 2.5.3) angegeben wurde, werden die in diesem Abschnitt aufgeführten Sätze für verschachtelte Variablenordnungen angegeben.

Wie in Abschnitt 4.3 beschrieben, erfolgt die Aufspaltung in einfachere Teilprobleme im Algorithmus bezüglich der für einen Rekursionsschritt ermittelten Top-Variablen v. Bezüglich dieser Top-Variablen werden Kofaktoren des BDDs des TLEBDDs R und des BDDs S durch Belegung der Top-Variable mit Booleschen Werten ermittelt. Diese werden bei den weiteren rekursiven Aufrufen des Algorithmus als Übergabeparameter verwendet. Die durch die rekursiven Aufrufe berechneten Teilergebnisse (R_0 bzw. R_1) werden anschließend zum Ergebnis eines Rekursionsschritts kombiniert. Im Algorithmus, der für verschachtelte Variablenordnungen angegeben ist, wird die Aufspaltung in Teilprobleme und die Berechnung des Resultats eines Rekursionsschritts bei einer Top-Variablen zur Kodierung eines Identitätsmusters, für eine Änderung des lokalen Zustands einer anderen Komponente als der für die gerade das relationale Produkt berechnet wird, für beide Variablen des Identitätsmusters gemeinsam ausgeführt. Die Korrektheit des dabei verwendeten Vorgehens wird in Satz 4.8 gezeigt.

Satz 4.8. *Gegeben seien das relationale Produkt bei Verwendung eines TLEBDDs für eine Komponente i, $\exists\vec{x}[\Delta(IdVar, R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}) \wedge S(\vec{x})]\{\vec{x}' \leftarrow \vec{x}\}$, wie es in Abschnitt 4.3 eingeführt wurde mit einer nichtleere Menge IdVar von Variablenpaaren, für die im TLEBDD für Komponente i implizit Identitätsmuster berücksichtigt werden. Sei außerdem v eine Boolesche Variable für aktuelle Zustände ($v \in X$) zur Kodierung des lokalen*

Zustands einer Komponente $j \neq i$, die daher für Komponente i eine Variable für Identitätsmuster ist und υ' sei die zur Variablen υ korrespondierende Variable für Nachfolgerzustände ($\upsilon' \in X'$) in Identitätsmustern und es gelte $(\upsilon,\ \upsilon') \in IdVar$. Dann kann die Berechnung des relationalen Produkts mit einem TLEBDD bezüglich der Belegung solcher Variablen υ und υ' mit Booleschen Werten folgendermaßen aufgespalten werden:

$$\begin{aligned}
&\exists\vec{x}[\Delta(IdVar, R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}) \wedge S(\vec{x})]\{\vec{x}' \leftarrow \vec{x}\}\\
&= \upsilon \cdot ((\exists\vec{x}(\Delta(IdVar', R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}) \wedge S(\vec{x})|_{\upsilon=1}))\{\vec{x}' \leftarrow \vec{x}\})\vee\\
&\overline{\upsilon} \cdot ((\exists\vec{x}(\Delta(IdVar', R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}) \wedge S(\vec{x})|_{\upsilon=0}))\{\vec{x}' \leftarrow \vec{x}\})
\end{aligned}$$

Dabei ist $IdVar'$ die Menge IdVar ohne das oben beschriebene Variablenpaar υ, υ', bezüglich dem die Aufspaltung durchgeführt wurde ($IdVar' = IdVar - \{(\upsilon,\ \upsilon')\}$).

Beweis.

$$\begin{aligned}
&\exists\vec{x}[\Delta(IdVar, R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}) \wedge S(\vec{x})]\{\vec{x}' \leftarrow \vec{x}\}\\
\overset{\substack{Shan.\\Exp.}}{=}&\exists\vec{x}(\upsilon \cdot \upsilon' \cdot (\Delta(IdVar, R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_i}\})|_{\upsilon=1,\ \upsilon'=1} \wedge S(\vec{x})|_{\upsilon=1,\ \upsilon'=1})\vee\\
&\upsilon \cdot \overline{\upsilon'} \cdot (\Delta(IdVar, R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_i}\})|_{\upsilon=1,\ \upsilon'=0} \wedge S(\vec{x})|_{\upsilon=1,\ \upsilon'=0})\vee\\
&\overline{\upsilon} \cdot \upsilon' \cdot (\Delta(IdVar, R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_i}\})|_{\upsilon=0,\ \upsilon'=1} \wedge S(\vec{x})|_{\upsilon=0,\ \upsilon'=1})\vee\\
&\overline{\upsilon} \cdot \overline{\upsilon'} \cdot (\Delta(IdVar, R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_i}\})|_{\upsilon=0,\ \upsilon'=0} \wedge S(\vec{x})|_{\upsilon=0,\ \upsilon'=0}))\{\vec{x}' \leftarrow \vec{x}\}\\
\overset{\substack{Ident.\\Pat.}}{=}&\exists\vec{x}(\upsilon \cdot \upsilon' \cdot (\Delta(IdVar - \{(\upsilon,\ \upsilon')\}, R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}) \wedge S(\vec{x})|_{\upsilon=1})\vee\\
&\upsilon \cdot \overline{\upsilon'} \cdot (0 \wedge S(\vec{x})|_{\upsilon=1}) \vee \overline{\upsilon} \cdot \upsilon' \cdot (0 \wedge S(\vec{x})|_{\upsilon=0})\vee\\
&\overline{\upsilon} \cdot \overline{\upsilon'} \cdot (\Delta(IdVar - \{(\upsilon,\ \upsilon')\}, R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}) \wedge S(\vec{x})|_{\upsilon=0}))\{\vec{x}' \leftarrow \vec{x}\}\\
=&\exists\vec{x}(\upsilon \cdot \upsilon' \cdot (\Delta(IdVar', R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}) \wedge S(\vec{x})|_{\upsilon=1})\vee\\
&\overline{\upsilon} \cdot \overline{\upsilon'} \cdot (\Delta(IdVar', R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}) \wedge S(\vec{x})|_{\upsilon=0}))\{\vec{x}' \leftarrow \vec{x}\}\\
\overset{\substack{\exists\vec{x}*}}{=}&(1 \cdot \upsilon' \cdot (\exists\vec{x}(\Delta(IdVar', R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}) \wedge S(\vec{x})|_{\upsilon=1}))\vee\\
&0 \cdot \upsilon' \cdot (\exists\vec{x}(\Delta(IdVar', R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}) \wedge S(\vec{x})|_{\upsilon=1}))\vee\\
&1 \cdot \overline{\upsilon'} \cdot (\exists\vec{x}(\Delta(IdVar', R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}) \wedge S(\vec{x})|_{\upsilon=0}))\vee\\
&0 \cdot \overline{\upsilon'} \cdot (\exists\vec{x}(\Delta(IdVar', R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}) \wedge S(\vec{x})|_{\upsilon=0})))\{\vec{x}' \leftarrow \vec{x}\}\\
\overset{\substack{Lemma\\2.42}}{=}&(\upsilon'\{\vec{x}' \leftarrow \vec{x}\}) \cdot ((\exists\vec{x}(\Delta(IdVar', R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}) \wedge S(\vec{x})|_{\upsilon=1}))\{\vec{x}' \leftarrow \vec{x}\})\vee\\
&(\overline{\upsilon'}\{\vec{x}' \leftarrow \vec{x}\}) \cdot ((\exists\vec{x}(\Delta(IdVar', R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}) \wedge S(\vec{x})|_{\upsilon=0}))\{\vec{x}' \leftarrow \vec{x}\})\\
=&\upsilon \cdot ((\exists\vec{x}(\Delta(IdVar', R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}) \wedge S(\vec{x})|_{\upsilon=1}))\{\vec{x}' \leftarrow \vec{x}\})\vee\\
&\overline{\upsilon} \cdot ((\exists\vec{x}(\Delta(IdVar', R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}) \wedge S(\vec{x})|_{\upsilon=0}))\{\vec{x}' \leftarrow \vec{x}\})
\end{aligned}$$

Durch Angabe von * wird oben eine Teilberechnung des Beweises von Lemma 3.1 (siehe Abschnitt 3.2), das für das relationale Produkt zwischen BDDs angegeben wurde, referenziert. Bei entsprechender Ausführung dieser Teilberechnung für TLEBDDs erhält man

das nach dem Gleichheitszeichen ($\overset{\exists\vec{x}}{\underset{*}{=}}$) aufgeführte Teilergebnis. Dabei ist hier nur der Fall $v \in X$ von Lemma 3.1 zu berücksichtigen. □

Da in einem TLEBDD keine Knoten für Identitätsmuster für Änderungen des lokalen Zustands einer anderen Komponente als der für die der TLEBDD gebaut wurde vorhanden sind, können im Algorithmus für Variablen solcher Identitätsmuster nur Knoten des BDDs S als Top-Variablen von Rekursionsschritten auftreten. Der BDD S besitzt nur wesentliche Variablen und entsprechende Knoten zur Kodierung von aktuelle Zuständen, weshalb die Top-Variable dafür eine Variable für aktuelle Zustände sein muss. Daher wird im Algorithmus die in Satz 4.8 bewiesene Aufspaltung in Teilprobleme für solche Identitätsmuster in Rekursionsschritten mit einer entsprechenden Variable für aktuelle Zustände als Top-Variable durchgeführt. Dazu werden für die Teilprobleme bei den rekursiven Aufrufen in den Zeilen 20 und 21 des Algorithmus, die in Satz 4.8 ermittelten Kofaktoren R (resultiert aus der Ausführung von Zeile 19 zuvor, es gilt $v \neq \mathit{mappedVar}$, da für die beschriebenen Identitätsmuster keine Knoten im BDD des TLEBDDs vorhanden sind), bzw. $S|_{v=1}$ oder $S|_{v=0}$ benutzt. Bei der Kombination der ermittelten Teilergebnisse ergibt sich bei gleichen Teilergebnissen sowohl für die in Satz 4.8 gezeigte Aufspaltung in Teilprobleme, als auch für die nachfolgend in Satz 4.9 bewiesene Aufspaltung, eines der Teilergebnisse als Rückgabewert eines Rekursionsschritts. Dies lässt sich an den angegebenen Aufspaltungen unter Berücksichtigung der Reduktionsregeln für BDDs leicht ablesen. Damit ist die in Algorithmus 12 in den Zeilen 22 und 23 angegebene Ermittlung des Rückgabewerts bei gleichen Teilergebnissen korrekt. Im weiteren Verlauf dieses Abschnitts wird mit Hilfe von Satz 4.8 und Satz 4.9 noch die Korrektheit der anderen im Algorithmus verwendeten Vorgehensweisen zur Resultatskombination beschrieben. Sind die berechneten Teilergebnisse nicht gleich, wird in Zeile 26 des Algorithmus zur Ermittlung des Resultats eines Rekursionsschritts für eine Top-Variable zur Kodierung eines Identitätsmusters ein neuer Knoten mit den ermittelten Teilergebnissen R_0 und R_1 als Nachfolgerknoten erzeugt. Damit ist für eine Eingabebelegung bei einer Belegung der Top-Variable v mit dem Booleschen Wert 1 das Teilergebnis R_1 und bei einer Belegung der Top-Variable v mit dem Booleschen Wert 0 das Teilergebnis R_0 zu berücksichtigen. Dies entspricht der in Satz 4.8 angegebenen Kombination der Teilergebnisse, bei der die Teilergebnisse jeweils für den Wert 0, bzw. den Wert 1 der Variable v zu berücksichtigen sind. Anschließend wird im Beweis zu Satz 4.9 die Korrektheit der Aufspaltung der Berechnung des relationalen Produkts in Teilprobleme bezüglich einer Top-Variablen v zur Kodierung des globalen Zustands oder des lokalen Zustands der Komponente, für die ein TLEBDD gebaut wurde, gezeigt. Im Beweis wird die Umkehrfunktion $\pi_i^{-1}(v)$ zur in Definition 4.1 in Abschnitt 4.1 eingeführten Zuordnungsfunktion π_i verwendet. Die Funktion π_i wurde dort zur Zuordnung der Variablenindizes von reduzierten Variablen, auf die diesen entsprechenden Variablenindizes für tatsächliche Variablen, verwendet. Aus Gründen der besseren Verständlichkeit bekommt die Umkehrfunktion hier als Eingabeparameter eine tatsächliche Variable übergeben und nicht den Variablenindex einer solchen Variable. Sie ermittelt dazu den Variablenindex der korrespondierenden reduzierten Variablen.

Satz 4.9. *Gegeben sei das relationale Produkt bei Verwendung eines TLEBDDs für eine Komponente i, $\exists\vec{x}[\Delta(IdVar, R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}) \wedge S(\vec{x})]\{\vec{x}' \leftarrow \vec{x}\}$, wie es in Abschnitt 4.3 eingeführt wurde. Sei außerdem υ eine Boolesche Variable zur Kodierung des globalen oder lokalen Zustands der Komponente i aus einer der Mengen X oder X'. Dann kann die Berechnung des relationalen Produkts mit einem TLEBDD bezüglich der Variable υ folgendermaßen aufgespalten werden:*

$$
\begin{aligned}
&\exists\vec{x}[\Delta(IdVar, R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}) \wedge S(\vec{x})]\{\vec{x}' \leftarrow \vec{x}\} = \\
&\begin{cases}
(\exists\vec{x}(\Delta(IdVar, R(\vec{y})|_{y_{\pi_i^{-1}(\upsilon)}=1}\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}) \wedge S(\vec{x})|_{\upsilon=1}))\{\vec{x}' \leftarrow \vec{x}\} \vee & \\
(\exists\vec{x}(\Delta(IdVar, R(\vec{y})|_{y_{\pi_i^{-1}(\upsilon)}=0}\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}) \wedge S(\vec{x})|_{\upsilon=0}))\{\vec{x}' \leftarrow \vec{x}\} & \upsilon \in X \\
\\
(\upsilon)\{\vec{x}' \leftarrow \vec{x}\} \cdot (\exists\vec{x}(\Delta(IdVar, R(\vec{y})|_{y_{\pi_i^{-1}(\upsilon)}=1}\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}) \wedge S(\vec{x}))) & \\
\{\vec{x}' \leftarrow \vec{x}\} \vee (\overline{\upsilon})\{\vec{x}' \leftarrow \vec{x}\} \cdot (\exists\vec{x}(\Delta(IdVar, R(\vec{y})|_{y_{\pi_i^{-1}(\upsilon)}=0}\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}) & \upsilon \in X' \\
\wedge S(\vec{x})))\{\vec{x}' \leftarrow \vec{x}\} &
\end{cases}
\end{aligned}
$$

Beweis.

$$
\begin{aligned}
&\exists\vec{x}[\Delta(IdVar, R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}) \wedge S(\vec{x})]\{\vec{x}' \leftarrow \vec{x}\} \\
\overset{\substack{Shan.\\Exp.}}{=\!=}\ &\exists\vec{x}(\upsilon \cdot (\Delta(IdVar, R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_i}\})|_{\upsilon=1} \wedge S(\vec{x})|_{\upsilon=1}) \vee \overline{\upsilon} \cdot (\Delta(IdVar, \\
&R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_i}\})|_{\upsilon=0} \wedge S(\vec{x})|_{\upsilon=0}))\{\vec{x}' \leftarrow \vec{x}\} \\
\overset{\substack{TLE-\\BDD}}{=\!=}\ &\exists\vec{x}(\upsilon \cdot (\Delta(IdVar, R(\vec{y})|_{y_{\pi_i^{-1}(\upsilon)}=1}\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}) \wedge S(\vec{x})|_{\upsilon=1}) \vee \overline{\upsilon} \cdot (\Delta(IdVar, \\
&R(\vec{y})|_{y_{\pi_i^{-1}(\upsilon)}=0}\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}) \wedge S(\vec{x})|_{\upsilon=0}))\{\vec{x}' \leftarrow \vec{x}\} \\
\overset{\substack{\exists\vec{x}\\*}}{=\!=}\ &
\begin{cases}
(1 \cdot (\exists\vec{x}(\Delta(IdVar, R(\vec{y})|_{y_{\pi_i^{-1}(\upsilon)}=1}\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}) \wedge S(\vec{x})|_{\upsilon=1}))\vee & \\
0 \cdot (\exists\vec{x}(\Delta(IdVar, R(\vec{y})|_{y_{\pi_i^{-1}(\upsilon)}=1}\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}) \wedge S(\vec{x})|_{\upsilon=1}))\vee & \\
1 \cdot (\exists\vec{x}(\Delta(IdVar, R(\vec{y})|_{y_{\pi_i^{-1}(\upsilon)}=0}\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}) \wedge S(\vec{x})|_{\upsilon=0}))\vee & \upsilon \in X \\
0 \cdot (\exists\vec{x}(\Delta(IdVar, R(\vec{y})|_{y_{\pi_i^{-1}(\upsilon)}=0}\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}) \wedge S(\vec{x})|_{\upsilon=0}))) & \\
\{\vec{x}' \leftarrow \vec{x}\} & \\
\\
(\upsilon \cdot (\exists\vec{x}(\Delta(IdVar, R(\vec{y})|_{y_{\pi_i^{-1}(\upsilon)}=1}\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}) \wedge S(\vec{x}))) & \\
\vee \overline{\upsilon} \cdot (\exists\vec{x}(\Delta(IdVar, R(\vec{y})|_{y_{\pi_i^{-1}(\upsilon)}=0}\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}) \wedge S(\vec{x})))) & \upsilon \in X' \\
\{\vec{x}' \leftarrow \vec{x}\} &
\end{cases}
\end{aligned}
$$

$$\overset{\substack{Lemma\\2.42}}{=} \begin{cases} (\exists\vec{x}(\Delta(IdVar, R(\vec{y})|_{y_{\pi_i^{-1}(\upsilon)}=1}\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}) \wedge S(\vec{x})|_{\upsilon=1}))\{\vec{x}' \leftarrow \vec{x}\} \vee & \\ (\exists\vec{x}(\Delta(IdVar, R(\vec{y})|_{y_{\pi_i^{-1}(\upsilon)}=0}\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}) \wedge S(\vec{x})|_{\upsilon=0}))\{\vec{x}' \leftarrow \vec{x}\} & \upsilon \in X \\ & \\ (\upsilon)\{\vec{x}' \leftarrow \vec{x}\} \cdot (\exists\vec{x}(\Delta(IdVar, R(\vec{y})|_{y_{\pi_i^{-1}(\upsilon)}=1}\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}) \wedge S(\vec{x}))) & \\ \{\vec{x}' \leftarrow \vec{x}\} \vee (\overline{\upsilon})\{\vec{x}' \leftarrow \vec{x}\} \cdot (\exists\vec{x}(\Delta(IdVar, R(\vec{y})|_{y_{\pi_i^{-1}(\upsilon)}=0}\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}) \wedge & \upsilon \in X' \\ S(\vec{x})))\{\vec{x}' \leftarrow \vec{x}\} & \end{cases}$$

Durch Angabe von * wird oben eine Teilberechnung des Beweises von Lemma 3.1 (siehe Abschnitt 3.2), das für das relationale Produkt zwischen BDDs angegeben wurde, referenziert. Bei entsprechender Ausführung dieser Teilberechnung für TLEBDDs erhält man das nach dem Gleichheitszeichen ($\overset{\exists\vec{x}}{\underset{*}{=}}$) aufgeführte Teilergebnis. □

In den beiden in Satz 4.9 bewiesenen Aufspaltungen in Teilprobleme, werden Kofaktoren des BDDs des TLEBDDs bezüglich der der Top-Variablen υ entsprechenden reduzierten Variablen in den Teilproblemen verwendet ($R(\vec{y})|_{y_{\pi_i^{-1}(\upsilon)}=1}$ bzw. $R(\vec{y})|_{y_{\pi_i^{-1}(\upsilon)}=0}$). Ist die Top-Variable υ eines Rekursionsschritts eine Variable zur Kodierung des globalen oder lokalen Zustands der Komponente für die der TLEBDD gebaut wurde, kann die Top-Variable im Algorithmus die entsprechende tatsächliche Variable zur reduzierten Variablen des Wurzelknotens des BDDs R des TLEBDDs, die tatsächliche Variable des Wurzelknotens des BDDs S oder beides sein. Wenn die tatsächliche Top-Variable die zur reduzierten Variablen des Wurzelknotens des BDDs R des TLEBDDs korrespondierende Variable ist, werden in Algorithmus 12 die BDDs der Kofaktoren $R_{w_1} = R|_{w=1}$ bzw. $R_{w_0} = R|_{w=0}$ als Übergabeparameter bei den rekursiven Aufrufen des Algorithmus in den Zeilen 20 bzw. 21 verwendet. Ist die reduzierte Variable des Wurzelknotens des BDDs R nicht die der tatsächlichen Variablen υ entsprechende reduzierte Variable, müssen keine neuen Kofaktoren des BDDs R gebildet werden. Daher wird der BDD R hier für beide rekursiven Aufrufe unverändert als Übergabeparameter verwendet und die im Algorithmus benutzten Kofaktoren des BDDs R entsprechen den im Satz bei den Teilproblemen verwendeten Kofaktoren.

Vom BDD S werden in Satz 4.9 bei einer Aufspaltung für eine Variable für aktuelle Zustände, Kofaktoren bezüglich dieser Variable in den Teilproblemen verwendet ($S(\vec{x})|_{\upsilon=1}$ bzw. $S(\vec{x})|_{\upsilon=0}$). Bei der Aufspaltung für eine Variable für Nachfolgerzustände, wird der BDDs S für beide Teilprobleme unverändert benutzt. Im Algorithmus werden für den BDD S immer Kofaktoren bezüglich der aktuellen Top-Variable eines Rekursionsschritts gebildet und auch bei den rekursiven Aufrufen für Teilprobleme verwendet. Ist die Top-Variable eine Variable für Nachfolgerzustände, entspricht ein Kofaktor des BDDs S für diese Variable dem BDD S selbst, da die mit dem BDD S repräsentierte Funktion nicht von Variablen für Nachfolgerzustände abhängig ist. Damit werden auch für den BDD S genau die in Satz 4.9 bei den Teilproblemen angegebenen Kofaktoren übergeben. Bei der Berechnung des Resultats eines Rekursionsschritts für eine Top-Variable zur Kodierung des globalen Zustands oder des lokalen Zustands der Komponente für die ein TLEBDD

gebaut wurde, werden vom Algorithmus zwei Fälle unterschieden. Ist die Top-Variable v eine Variable für aktuelle Zustände, wird die Veroderung der Teilergebnisse R_0 und R_1 als Ergebnis eines solchen Rekursionsschritts zurückgegeben (siehe Zeile 29 in Algorithmus 12). Andernfalls wird ein neuer Knoten für die der Top-Variablen v für Nachfolgerzustände entsprechende Variable für aktuelle Zustände, mit den BDDs der ermittelten Teilresultate R_0 und R_1 als Nachfolgern, erzeugt. Diese Resultatskombination entspricht für beide in Satz 4.9 angegebenen Fälle der dort verwendeten Vereinigung der Teilergebnisse. Die bei einer Top-Variablen für Nachfolgerzustände mit Hilfe der Umbenennungsoperation durchzuführende Umbenennung in die entsprechende Variable für aktuelle Zustände ist dabei in Satz 4.9 durch $(v)\{\vec{x}' \leftarrow \vec{x}\}$ bzw. $(\overline{v})\{\vec{x}' \leftarrow \vec{x}\}$ angegeben. Das Vorgehen des Algorithmus bei der rekursiven Aufspaltung in Teilprobleme und die Kombination der resultierenden Teilergebnisse entspricht damit genau den in Satz 4.8 und Satz 4.9 als korrekt bewiesenen Aufspaltungen in Teilprobleme. Daher ist diese Vorgehensweise des Algorithmus bei der Aufspaltung in Teilprobleme korrekt und wird rekursiv bis zum Erreichen eines Terminalfalls durchgeführt. Die Korrektheit der verwendeten Terminalfälle wird im nachfolgenden Abschnitt 4.5 gezeigt.

4.5 Korrektheit der Terminalfälle

Die im in Abschnitt 4.3 vorgestellten Algorithmus zur kombinierten Berechnung des relationalen Produkts mit einem TLEBDD verwendeten Terminalfälle und deren Rückgabewerte sind in der in Algorithmus 13 aufgeführten Funktion *TerminalCase()* angegeben. Als erste Terminalfälle werden dort die Terminalfälle, bei denen der Wurzelknoten des BDDs R oder der Wurzelknoten des BDD S ein Terminalknoten für den Booleschen Wert 0 ist, abgefragt. Entspricht einer dieser BDDs dem Terminalknoten für den Booleschen Wert 0, wird der Wert 0 als Rückgabewert zurückgegeben (siehe Zeilen 2 und 3 in Algorithmus 13). Diese Terminalfälle wurden bereits im in Abschnitt 3.1 vorgestellten Algorithmus zur kombinierten Berechnung des relationalen Produkts zwischen BDDs verwendet. Wie in Abschnitt 4.3 beschrieben, ändert sich der Rückgabewert dieser Terminalfälle auch durch die im TLEBDD fehlenden und möglicherweise noch zu berücksichtigenden Identitätsmuster nicht. Bei Expansion des TLEBDDs in einen korrespondierenden BDD wäre die gleiche Teilberechnung wie im relationalem Produkt zwischen BDDs durchzuführen.

Weitere Terminalfälle treten beim Algorithmus zur kombinierten Berechnung des relationalen Produkts mit einem TLEBDD auf, wenn der Wurzelknoten des BDDs R des TLEBDDs ein Terminalknoten für den Booleschen Wert 1 ist. In einem TLEBDD für eine Komponente i sind für Identitätsmuster zur Kodierung von Änderungen des lokalen Zustands anderer Komponenten eines Systems keine Knoten vorhanden. Für die entsprechenden Variablen wird implizit die Interpretation als Teil eines Identitätsmusters verwendet. Der BDD R ist dagegen ein gewöhnlicher BDD, auf den die für diese verwendeten Reduktionsregeln angewandt werden. Ist der Wurzelknoten des BDDs R in einem Rekursionsschritt ein Terminalknoten für den Booleschen Wert 1, können in diesem Rekursionsschritt noch zu evaluierende Variablen der Variablenordnung wie in Abschnitt 4.3 beschrieben, die Interpretation als Variable eines Identitätsmusters oder die in ge-

wöhnlichen BDDs verwendete Interpretation besitzen. Aus den verschiedenen Interpretationen können unterschiedliche Ergebnisse bei der Berechnung des relationalen Produkts resultieren, wodurch die Berechnung nicht ohne weitere rekursive Aufrufe zur Ermittlung der für noch zu evaluierende Variablen zu verwendenden Interpretation terminiert werden kann. Dies wurde bei der Auswahl der drei weiteren in Algorithmus 13 aufgeführten Terminalfälle beachtet. Keine Auswirkung haben noch zu evaluierende Variablen beider Interpretationen, wenn in einem Rekursionsschritt auch der Wurzelknoten des BDDs S ein Terminalknoten für den Booleschen Wert 1 ist. Dies wird im Beweis zu Satz 4.10 gezeigt.

Satz 4.10. *Gegeben seien das relationale Produkt bei Verwendung eines TLEBDDs für eine Komponente i, $\exists\vec{x}[\Delta(IdVar, R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}) \wedge S(\vec{x})]\{\vec{x}' \leftarrow \vec{x}\}$, wie es in Abschnitt 4.3 eingeführt wurde. Seien außerdem der BDD $R(\vec{y})$ des TLEBDDs, sowie der BDD $S(\vec{x})$ Terminalknoten für den Booleschen Wert 1. Dann ist das Ergebnis der Berechnung des relationalen Produkts mit einem TLEBDD der BDD für den Booleschen Wert 1.*

Beweis. Die Operation $\Delta(IdVar, 1)$ fügt, wie in Definition 4.4 beschrieben, für alle in der Menge *IdVar* enthaltenen Variablenpaare Identitätstransformationen in den der Operation im zweiten Argument übergebenen BDD ein. Das Einfügen von Identitätstransformationen in einen BDD für den Terminalknoten für den Booleschen Wert 1 verringert beim BDD des Resultats die Menge der Eingabebelegungen, die auf den Booleschen Wert 1 auswerten. Im nachfolgenden Beweis wird das Resultat des Terminalfalls unter der Annahme, dass für alle korrespondierenden Variablenpaare (aus X und X') Identitätstransformationen zu berücksichtigen sind bewiesen. Da bei einer kleineren Menge *IdVar* nur weniger Beschränkungen in den BDD mit dem Terminalknoten für den Booleschen Wert 1 einzufügen sind, wertet der Resultat-BDD für eine größere Menge an Eingabebelegungen auf den Wert 1 aus. Die Wirkung der vorhandenen Identitätstransformationen wird dabei im Beweis in der mit *Ident. Trans* gekennzeichneten Zeile verwendet. Der On-Set des Resultats würde sich bei weniger vorhandenen Identitätstransformationen vergrößern. Damit gilt der Satz auch für alle kleineren Mengen *IdVar*.

$$
\begin{aligned}
&\exists\vec{x}[\Delta(IdVar, R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}) \wedge S(\vec{x})]\{\vec{x}' \leftarrow \vec{x}\} \\
\overset{\substack{R(\vec{y})=1\\ S(\vec{x})=1}}{=}\ &(\exists\vec{x}[\Delta(IdVar, (1)\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}) \wedge 1])\{\vec{x}' \leftarrow \vec{x}\} = (\exists\vec{x}[\Delta(IdVar, 1) \wedge 1])\{\vec{x}' \leftarrow \vec{x}\} \\
\overset{\substack{Shannon\\ Exp.}}{=}\ &\exists\vec{x}(\overline{x_1}\overline{x'_1} \ldots \overline{x_n}\overline{x'_n}\Delta(IdVar, 1)|_{x_1=0,\, x'_1=0,\, \ldots,\, x_n=0,\, x'_n=0} \\
&\quad \vee x_1\overline{x'_1} \ldots \overline{x_n}\overline{x'_n}\Delta(IdVar, 1)|_{x_1=1,\, x'_1=0,\, \ldots,\, x_n=0,\, x'_n=0} \vee \ldots \\
&\quad \vee x_1x'_1 \ldots x_nx'_n\Delta(IdVar, 1)|_{x_1=1,\, x'_1=1,\, \ldots,\, x_n=1,\, x'_n=1})\{\vec{x}' \leftarrow \vec{x}\} \\
\overset{\exists\vec{x}}{=}\ &(\overline{x'_1} \ldots \overline{x'_n}\Delta(IdVar, 1)|_{x_1=0,\, x'_1=0,\, \ldots,\, x_n=0,\, x'_n=0} \\
&\quad \vee x'_1 \ldots \overline{x'_n}\Delta(IdVar, 1)|_{x_1=1,\, x'_1=0,\, \ldots,\, x_n=0,\, x'_n=0} \vee \ldots \\
&\quad \vee x'_1 \ldots x'_n\Delta(IdVar, 1)|_{x_1=1,\, x'_1=1,\, \ldots,\, x_n=1,\, x'_n=1})\{\vec{x}' \leftarrow \vec{x}\}
\end{aligned}
$$

$$\begin{aligned}
&\overset{Ident.}{\underset{Trans}{=}}(\overline{x'_1}\,...\,\overline{x'_n}\Delta(IdVar,1)|_{x_1=0,\,x'_1=0,\,...,\,x_n=0,\,x'_n=0}\\
&\quad\vee x'_1\overline{x'_2}\,...\,\overline{x'_n}\Delta(IdVar,1)|_{x_1=1,\,x'_1=1,\,x_2=0,\,x'_2=0,\,...,\,x_n=0,\,x'_n=0}\\
&\quad\vee\,...\,\vee x'_1\,...\,x'_n\Delta(IdVar,1)|_{x_1=1,\,x'_1=1,\,...,\,x_n=1,\,x'_n=1})\{\vec{x}'\leftarrow\vec{x}\}\\
=&(\overline{x'_1}\,...\,\overline{x'_n}\cdot 1\vee x'_1\overline{x'_2}\,...\,\overline{x'_n}\cdot 1\vee\,...\,\vee x'_1\,...\,x'_n\cdot 1)\{\vec{x}'\leftarrow\vec{x}\}=(1)\{\vec{x}'\leftarrow\vec{x}\}=1
\end{aligned}$$

□

Neben dem in Satz 4.10 behandelten Terminalfall werden noch zwei weitere Terminalfälle verwendet, die auftreten können wenn der BDD R dem Terminalknoten für den Booleschen Wert 1 entspricht. Beide treten ein, wenn zusätzlich in einem Rekursionsschritt ab der Variablen des Wurzelknotens des BDDs S nur noch Variablen mit einer Interpretation in der verwendeten Variablenordnung vorkommen. Im nachfolgenden Satz 4.11 wird die Korrektheit des Terminalfalls gezeigt, der eintritt wenn zusätzlich in der Variablenordnung ab der Variablen des Wurzelknotens des BDDs S nur noch Variablen zur Kodierung von globalen oder lokalen Variablen der Komponente, für die der TLEBDD gebaut wurde, vorkommen. Dieser und auch der Terminalfall, dessen Korrektheit anschließend in Satz 4.12 gezeigt wird, können in der Funktion *TerminalCase()* nur eintreten, wenn der BDD S nicht dem Terminalknoten für den Booleschen Wert 0 entspricht. Andernfalls wären bereits die zuvor abgefragten Terminalfälle erfüllt.

Satz 4.11. *Gegeben sei das relationale Produkt bei Verwendung eines TLEBDDs für eine Komponente i, $\exists\vec{x}[\Delta(IdVar,R(\vec{y})\{\vec{y}\leftarrow\vec{x}_{\pi_i}\})\wedge S(\vec{x})]\{\vec{x}'\leftarrow\vec{x}\}$, wie es in Abschnitt 4.3 eingeführt wurde und eine verschachtelte Variablenordnung für die tatsächlichen Variablen. Entspricht der BDD R dem Terminalknoten für den Booleschen Wert 1, der BDD S ist ungleich dem Terminalknoten für den Booleschen Wert 0 und folgen ab der Variablen des Wurzelknotens des BDDs S in der gegebenen Variablenordnung der tatsächlichen Variablen nur noch Variablen zur Kodierung des globalen Zustands oder des lokalen Zustands der Komponente i, dann ist das Ergebnis der Berechnung des relationalen Produkts $\exists\vec{x}[\Delta(IdVar,R(\vec{y})\{\vec{y}\leftarrow\vec{x}_{\pi_i}\})\wedge S(\vec{x})]\{\vec{x}'\leftarrow\vec{x}\}$ der Terminalknoten für den Booleschen Wert 1.*

Beweis.

$$\begin{aligned}
&\exists\vec{x}[\Delta(IdVar,R(\vec{y})\{\vec{y}\leftarrow\vec{x}_{\pi_i}\})\wedge S(\vec{x})]\{\vec{x}'\leftarrow\vec{x}\}\\
*\;&\overset{R=1}{\underset{IdVar=\emptyset}{=}}\exists\vec{x}[\Delta(\emptyset,1)\wedge S(\vec{x})]\{\vec{x}'\leftarrow\vec{x}\}\\
=&\exists\vec{x}[1\wedge S(\vec{x})]\{\vec{x}'\leftarrow\vec{x}\}\overset{Beweis}{\underset{Satz\ 3.3}{=}}1
\end{aligned}$$

Da der BDD S nicht der Terminalknoten für den Booleschen Wert 0 ist, kann im Beweis der entsprechende Fall von Satz 3.3 zur Ermittlung des Booleschen Werts 1 als Resultat angewandt werden. Bei den zuvor im Beweis angegebenen Umformungen wurde die Menge $IdVar$ auf die leere Menge gesetzt ($IdVar=\emptyset$, im Beweis mit * gekennzeichnet). Dies ist korrekt, da ab der Variablen des Wurzelknotens des BDDs S in der gegebenen

Variablenordnung der tatsächlichen Variablen wie im Satz oben angegeben nur noch Variablen zur Kodierung des globalen Zustands oder Variablen zur Kodierung des lokalen Zustands der Komponente i vorkommen. Für solche Variablen sind bei einem TLEBDD für Komponente i keine Identitätsmuster einzufügen. Hinsichtlich vor der Variablen des Wurzelknotens des BDDs S in der Variablenordnung vorkommender Variablen, sind im BDD S und im TLEBDD ($R = 1$) keine Knoten für diese vorhanden. Bei nicht vorhandenen Knoten für ein Variablenpaar aus Variable für aktuelle Zustände und Variable für Nachfolgerzustände aus dieser Menge, ergibt sich das gleiche Ergebnis für das relationale Produkt bei einem in der Transitionsrelation für das Variablenpaar zu berücksichtigenden Identitätsmuster und bei Verwendung der bei üblichen BDDs aufgrund der Reduktionsregeln bei fehlenden Knoten zu berücksichtigenden Interpretation für das Variablenpaar in der Transitionsrelation. □

Damit ist der korrekte Rückgabewert für diesen Terminalfall der BDD mit dem Terminalknoten für den Wert 1, der für diesen auch in der Funktion *TerminalCase()* verwendet wird.

Ein weiterer Terminalfall, der den BDD S als Rückgabewert besitzt, liegt vor wenn der Wurzelknoten des BDDs R ein Terminalknoten für den Booleschen Wert 1 ist und in der Variablenordnung ab der Variablen des Wurzelknotens des BDDs S nur noch Variablen mit der Interpretation als Teil eines Identitätsmusters vorkommen. Die Korrektheit dieses Terminalfalls und damit auch des für diesen in der Funktion *TerminalCase()* verwendeten Rückgabewerts wird in Satz 4.12 gezeigt.

Satz 4.12. *Gegeben seien das relationale Produkt bei Verwendung eines TLEBDDs für eine Komponente i, $\exists\vec{x}[\Delta(IdVar, R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}) \wedge S(\vec{x})]\{\vec{x}' \leftarrow \vec{x}\}$, wie es in Abschnitt 4.3 eingeführt wurde und eine verschachtelte Variablenordnung für die tatsächlichen Variablen. Entspricht der BDD R dem Terminalknoten für den Booleschen Wert* 1*, der BDD S ist kein Terminalknoten und folgen ab der Variablen des Wurzelknotens des BDDs S in der Variablenordnung nur noch Variablenpaare für Identitätsmuster zur Kodierung der Änderungen des lokalen Zustands der anderen Komponenten $j \neq i$, ist das Ergebnis der Berechnung des relationalen Produkts $\exists\vec{x}[\Delta(IdVar, R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}) \wedge S(\vec{x})]\{\vec{x}' \leftarrow \vec{x}\}$ der BDD S.*

Beweis. Sei v_i die tatsächliche Variable des Wurzelknotens des BDDs S, $\{v_i, v_{i+1}, ..., v_n\}$ die Menge der ab der Variablen v_i in der Variablenordnung vorkommenden Variablen für aktuelle Zustände und $\{v'_i, v'_{i+1}, ..., v'_n\}$ die Menge der jeweils zu diesen Variablen korrespondierenden noch vorkommenden Variablen für Nachfolgerzustände, wobei die Variablen mit dem gleichen Variablenindex Variablenpaare zur Kodierung von Identitätsmustern bilden. Da inklusive des Variablenpaares mit der Variablen des Wurzelknotens des BDDs S in der Variablenordnung nur noch Variablenpaare zur Kodierung von Identitätsmustern vorkommen, sind alle diese Variablenpaare in der Menge $IdVar$ enthalten. Es

gilt für die Berechnung des relationalen Produkts:

$$
\begin{aligned}
& \exists \vec{x}[\Delta(IdVar, R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}) \wedge S(\vec{x})]\{\vec{x}' \leftarrow \vec{x}\} \\
\overset{R=1}{=} & \exists \vec{x}[\Delta(IdVar, 1) \wedge S(\vec{x})]\{\vec{x}' \leftarrow \vec{x}\} \\
\overset{Satz}{\underset{4.8}{=}} & v_i \cdot (\exists \vec{x}(\Delta(IdVar - \{(v_i, v_i')\}, 1) \wedge S(\vec{x})|_{v_i=1})\{\vec{x}' \leftarrow \vec{x}\} \vee \\
& \overline{v_i} \cdot (\exists \vec{x}(\Delta(IdVar - \{(v_i, v_i')\}, 1) \wedge S(\vec{x})|_{v_i=0})\{\vec{x}' \leftarrow \vec{x}\} \\
\overset{Satz}{\underset{4.8}{=}} & v_i \cdot (v_{i+1} \cdot (\exists \vec{x}(\Delta(IdVar - \{(v_i, v_i')\} - \{(v_{i+1}, v_{i+1}')\}, 1) \wedge \\
& S(\vec{x})|_{v_i=1,\, v_{i+1}=1})\{\vec{x}' \leftarrow \vec{x}\}) \vee \overline{v_{i+1}} \cdot (\exists \vec{x}(\Delta(IdVar - \\
& \{(v_i, v_i')\} - \{(v_{i+1}, v_{i+1}')\}, 1) \wedge S(\vec{x})|_{v_i=1,\, v_{i+1}=0})\{\vec{x}' \leftarrow \vec{x}\})) \vee \\
& \overline{v_i} \cdot (v_{i+1} \cdot (\exists \vec{x}(\Delta(IdVar - \{(v_i, v_i')\} - \{(v_{i+1}, v_{i+1}')\}, 1) \wedge \\
& S(\vec{x})|_{v_i=1,\, v_{i+1}=1})\{\vec{x}' \leftarrow \vec{x}\}) \vee \overline{v_{i+1}} \cdot (\exists \vec{x}(\Delta(IdVar - \\
& \{(v_i, v_i')\} - \{(v_{i+1}, v_{i+1}')\}, 1) \wedge S(\vec{x})|_{v_i=1,\, v_{i+1}=0})\{\vec{x}' \leftarrow \vec{x}\})) \\
\overset{Satz}{\underset{4.8}{=}} & \ldots \\
* \overset{Satz}{\underset{4.8}{=}} & v_i\, v_{i+1} \ldots v_n \cdot ((\exists \vec{x}(\Delta(\emptyset, 1) \wedge S(\vec{x})|_{v_i=1,\, v_{i+1}=1,\, \ldots,\, v_n=1}))\{\vec{x}' \leftarrow \vec{x}\}) \vee \\
& v_i\, v_{i+1} \ldots \overline{v_n} \cdot ((\exists \vec{x}(\Delta(\emptyset, 1) \wedge S(\vec{x})|_{v_i=1,\, v_{i+1}=1,\, \ldots,\, v_n=0}))\{\vec{x}' \leftarrow \vec{x}\}) \vee \ldots \\
& \vee \overline{v_i}\, \overline{v_{i+1}} \ldots \overline{v_n} \cdot ((\exists \vec{x}(\Delta(\emptyset, 1) \wedge S(\vec{x})|_{v_i=0,\, v_{i+1}=0,\, \ldots,\, v_n=0}))\{\vec{x}' \leftarrow \vec{x}\}) \\
= & v_i\, v_{i+1} \ldots v_n \cdot ((\exists \vec{x}(S(\vec{x})|_{v_i=1,\, v_{i+1}=1,\, \ldots,\, v_n=1}))\{\vec{x}' \leftarrow \vec{x}\}) \vee \\
& v_i\, v_{i+1} \ldots \overline{v_n} \cdot ((\exists \vec{x}(S(\vec{x})|_{v_i=1,\, v_{i+1}=1,\, \ldots,\, v_n=0}))\{\vec{x}' \leftarrow \vec{x}\}) \vee \ldots \\
& \vee \overline{v_i}\, \overline{v_{i+1}} \ldots \overline{v_n} \cdot ((\exists \vec{x}(S(\vec{x})|_{v_i=0,\, v_{i+1}=0,\, \ldots,\, v_n=0}))\{\vec{x}' \leftarrow \vec{x}\}) \\
\overset{S(\vec{x})|_{..} \in \{0,1\}}{=} & v_i\, v_{i+1} \ldots v_n \cdot S(\vec{x})|_{v_i=1,\, v_{i+1}=1,\, \ldots,\, v_n=1} \vee \\
& v_i\, v_{i+1} \ldots \overline{v_n} \cdot S(\vec{x})|_{v_i=1,\, v_{i+1}=1,\, \ldots,\, v_n=0} \vee \ldots \vee \\
& \overline{v_i}\, \overline{v_{i+1}} \ldots \overline{v_n} \cdot S(\vec{x})|_{v_i=0,\, v_{i+1}=0,\, \ldots,\, v_n=0} = S(\vec{x})
\end{aligned}
$$

Bei den zuvor im Beweis angegebenen Umformungen wurde die Menge $IdVar$ auf die leere Menge gesetzt ($IdVar = \emptyset$) und die entsprechende Umformung wurde mit * gekennzeichnet. Dies ist korrekt, da alle nach der Variablen v_i in der Variablenordnung vorkommenden Variablen in den Umformungen zuvor berücksichtigt wurden und im BDD S und im TLEBDD ($R = 1$) für vor der Variablen v_i des Wurzelknotens des BDDs S in der Variablenordnung vorkommende Variablen keine Knoten vorhanden sind. Bei nicht vorhandenen Knoten für ein Variablenpaar aus Variable für aktuelle Zustände und Variable für Nachfolgerzustände aus dieser Menge, ergibt sich das gleiche Ergebnis für das relationale Produkt bei einem in der Transitionsrelation für das Variablenpaar zu berücksichtigenden Identitätsmuster und bei Verwendung der bei üblichen BDDs aufgrund der Reduktionsregeln bei fehlenden Knoten zu berücksichtigenden Interpretation für das Variablenpaar in der Transitionsrelation. □

4.6 Abgrenzung zu verwandten Arbeiten

In diesem Abschnitt werden die in diesem Kapitel eingeführte Datenstruktur des TLEBDDs und der ebenfalls vorgestellte Algorithmus zur kombinierten Berechnung des relationalen Produkts mit einem TLEBDD mit verwandten Arbeiten verglichen. Weitere Ansätze zur Repräsentation der Transitionsrelation von Systemen oder Verfahren zur effizienten Bildberechnung wurden in Abschnitt 2.5 vorgestellt. TLEBDDs sind eine effiziente Datenstruktur zur komponentenweise partitionierten Repräsentation der Transitionsrelation asynchroner nebenläufiger System mit Transitionslokalität. Partitionierte Transitionsrelationen aus BDDs, die zur Reduktion des Speicherbedarfs einer Repräsentation der Transitionsrelation durch einen monolithischen BDD entwickelt wurden, wurden in Abschnitt 2.5.2 der vorliegenden Arbeit vorgestellt. Die implizite Repräsentation von Identitätstransformationen und die Verwendung einer reduzierten Variablenmenge bei TLEBDDs, kann im Vergleich zur Benutzung von BDDs zu einer weiteren großen Reduktionen des Speicherbedarfs führen. Die Benutzung einer reduzierten Variablenmenge und das Kodieren der Transitionen verschiedener Komponenten über diesen Variablen, ermöglicht es isomorphe Teilgraphen bei der Repräsentation von Transitionen verschiedener Komponenten zu erhalten. Dadurch müssen diese von gängigen BDD-Programmpaketen nur einmal abgespeichert werden. Dies kann den Speicherbedarf für die Transitionsrelation von Systemen, bei denen viele Ähnlichkeiten zwischen TLEBDDs zur Repräsentation von Partitionen der Transitionsrelation vorhanden sind, stark verringern. Dadurch können selbst im Vergleich mit einer partitionierten Transitionsrelation aus BDDs, bei der in den BDDs für Partitionen von Komponenten ebenfalls keine Identitätsmuster für Änderungen des lokalen Zustands anderer Komponenten abgespeichert werden, noch weitere große Verbesserungen des Speicherbedarfs erzielt werden (siehe experimentelle Ergebnisse in Abschnitt 8.4).

Weitere Ansätze, die ebenfalls versuchen die abzuspeichernden Identitätstransformationen in Repräsentationen der Transitionsrelation zu verringern, wurden in Abschnitt 2.5.4 vorgestellt. Dort wurden für verschiedene Typen von Entscheidungsdiagrammen (BDDs, MxDs, MDDs, erweiterte MDDs und LDDs) Strategien dazu beschrieben. Das Ziel aller dort präsentierten Verfahren ist wie auch bei TLEBDDs das Verhindern des expliziten Abspeicherns von Identitätstransformationen, ohne die Wirkung dieser Identitätstransformationen bei Operationen mit den Datenstrukturen zu verlieren. Bei all diesen Ansätzen findet aber keine zusätzliche Verwendung einer reduzierten Variablenmenge und die Abbildung der ursprünglich verwendeten tatsächlichen Variablen auf diese Variablenmenge wie bei TLEBDDs statt. Durch die Verwendung einer reduzierten Variablenmenge können bei TLEBDDs weitere große Verringerungen des Speicherbedarfs bei der Repräsentation von Transitionsrelationen erzielt werden.

Verfahren, die wie TLEBDDs versuchen das Vorhandensein von bis auf Variablenmarkierungen von Knoten isomorpher Teilgraphen verstärkt auszunutzen, wurden in Abschnitt 2.5.5 vorgestellt. Bei keinem dieser Verfahren wird, wie bei TLEBDDs, eine zusätzliche Entfernung von Identitätstransformationen durchgeführt. Ein Umbenennen der Eingabevariablen zur Ermöglichung des Ausnutzens von bis auf Variablenmarkierungen von Knoten isomorpher Teilgraphen, wurde bei den in [99] vorgestellten geordneten Funk-

tionsvorlagen für synchrone Systeme benutzt. Dort wird ebenfalls eine Zuordnungsliste zur Zuordnung der Variablen nach der Umbenennung zu den ursprünglichen Variablen verwendet. Es werden aber keine Identitätstransformationen für Änderungen des lokalen Zustands anderer Komponenten, wie sie in asynchronen Systemen vorhanden sind, beachtet. Es findet nur eine Umbenennung der Eingabevariablen statt. Ein weiteres Verfahren, bei dem ebenfalls eine Umbenennung von Variablen durchgeführt wird, ist das in [198] für die Verwendung konjunktiv partitionierter Transitionsrelationen vorgestellte Verfahren. Dort wird bei Verwendung von Early Quantification eine Umbenennung der Variablen von Knoten auf nicht mehr benötigte Variablen durchgeführt, um das Potential zum Ausnutzen isomorpher Teile zu vergrößern. Neben dem Entwurf dieses Verfahrens für konjunktiv partitionierte Transitionsrelationen, können dort Reduktionen des Speicherbedarfs erst während der Bildberechnung und nicht wie bei TLEBDDs schon beim Aufbau der Transitionsrelation erzielt werden.

Eine Abgrenzung der kombinierten Berechnung des relationalen Produkts für BDDs zu verwandten Ansätzen wurde in Abschnitt 3.5 angegeben. Die dort aufgezeigten Unterschiede zu anderen Ansätzen zur Bildberechnung gelten auch für den in Abschnitt 4.3 vorgestellten Algorithmus zur kombinierten Berechnung des relationalen Produkts mit TLEBDDs. Zusätzlich wird im Algorithmus für TLEBDDs während der rekursiven Berechnung des relationalen Produkts noch die Umbenennung der reduzierten Variablen des TLEBDDs in entsprechende tatsächliche Variablen durchgeführt. Außerdem findet eine implizite Berücksichtigung der im TLEBDD fehlenden Identitätsmuster statt und es werden für die Verwendung von TLEBDDs optimierte Terminalfälle verwendet. Ein Verfahren zur Bildberechnung für eine um Identitätsmuster reduzierte Transitionsrelation und BDDs wurde im in Abschnitt 2.5.4 präsentierten Ansatz aus [11] vorgestellt. Dort wird aber kein Algorithmus präsentiert, der alle Operationen des relationalen Produkts gemeinsam berechnet. Stattdessen wird ein Verfahren präsentiert, bei dem zuerst eine Umbenennung von Variablen durchgeführt wird. Anschließend wird ein gewöhnlicher Algorithmus zur Bildberechnung verwendet. Da hier bei der Transitionsrelation nur eine Reduktion um Identitätsmuster erfolgt, wird auch keine Umbenennung von Eingabevariablen, wie sie im Algorithmus für TLEBDDs von reduzierten Variablen auf tatsächliche Variablen gemacht wird, durchgeführt. Für die geordneten Funktionsvorlagen, bei denen ebenfalls eine Umbenennung der Eingabevariablen erfolgt, werden in [99] auch Verfahren zur Berechnung Boolescher Operationen damit angegeben. Diese basieren aber auf einer Expansion der Funktionsvorlagen in einen gewöhnlichen BDD und der anschließenden Ausführung der üblichen Booleschen Operationen zwischen BDDs. Bei Benutzung des in 4.3 präsentierten Algorithmus zur kombinierten Berechnung des relationalen Produkts mit einem TLEBDD können der Zeitaufwand zur Konstruktion des unreduzierten BDDs und der Speicherbedarf für diesen BDD eingespart werden.

5 ETLEBDDs

In diesem Kapitel werden *erweiterte Transitionslokalitätsausnutzende binäre Entscheidungsdiagramme (engl. extended transition locality exploiting BDDs, ETLEBDDs)* vorgestellt. Ein ETLEBDD ist eine Datenstruktur, die zur Repräsentation der Transitionsrelation von asynchronen nebenläufigen Systemen mit Transitionslokalität (siehe Abschnitt 2.3) benutzt werden kann.

Im Gegensatz zu den in Kapitel 4 eingeführten TLEBDDs, in denen jeweils nur Transitionen einer Komponente eines solchen Systems gespeichert werden, können mit einem ETLEBDD Transitionen verschiedener Komponenten gemeinsam repräsentiert werden. Für die von verschiedenen Komponenten gespeicherten Transitionen muß dabei gelten, dass die Änderungen des globalen Zustands und die des lokalen Zustands der jeweiligen ausführenden Komponente bei Transitionen mit dem gleichen BDD des ETLEBDDs repräsentiert werden können. Neben der Datenstruktur ETLEBDD wird in diesem Kapitel unter anderem auch ein Algorithmus zur kombinierten Berechnung des relationalen Produkts mit ETLEBDDs vorgestellt (siehe Abschnitt 5.3). Dieser ermöglicht für Komponenten, für die in einem ETLEBDD Transitionen gespeichert sind, die simultane und einmalige Ausführung von gleich durchzuführenden Teilberechnungen des relationalen Produkts. Dadurch können große Laufzeitverbesserungen erzielt werden (siehe Ergebnisse von Verifikationsexperimenten in Kapitel 8).

5.1 Die Datenstruktur ETLEBDD

Mit einem TLEBDD können Transitionen einer Komponente eines asynchronen nebenläufigen Systems mit Transitionslokalität repräsentiert werden (siehe Kapitel 4). Erweiterte Transitionslokalitätsausnutzende binäre Entscheidungsdiagramme (engl. extended transition locality exploiting BDDs, ETLEBDDs) sind eine Erweiterung von TLEBDDs, durch die die gemeinsame Speicherung von Mengen von Transitionen verschiedener Komponenten eines solchen Systems ermöglicht wird. Bei asynchronen nebenläufigen Systemen mit Transitionslokalität und der Verwendung einer komponentenweise partitionierten Transitionsrelation aus BDDs, weisen BDDs zur Repräsentation von Transitionen verschiedener Komponenten häufig strukturelle Ähnlichkeiten auf. Vor allem bei Systemen aus replizierten Komponenten ist dies oft bei der Repräsentation der Änderungen des globalen Zustands und der des lokalen Zustands der Komponenten der Fall. Die meisten BDD-Programmpakete speichern isomorphe Teilgraphen nur einmal ab. Da die lokalen Zustände verschiedener Komponenten aber über unterschiedlichen Booleschen Variablen kodiert werden, liegen hier oft strukturell gleiche Teilgraphen mit Knoten für verschiedene Variablenmengen vor. Zusätzlich unterscheiden sich die Variablenmengen zur Ko-

dierung von Identitätstransformationen. Der Grund ist die Benutzung unterschiedlicher Variablenmengen für Knoten für Identitätstransformationen für Änderungen des lokalen Zustands von Komponenten, für die ein BDD für eine Partition der Transitionsrelation nicht gebaut wurde. Daher können mit partitionierten Transitionsrelationen aus BDDs solche bei der komponentenweisen Repräsentation von Transitionsrelationen vorhandenen strukturelle Ähnlichkeiten von BDD-Programmpaketen nicht ausgenutzt werden, weil dort keine isomorphen Teilgraphen vorliegen. TLEBDDs wurden entwickelt, um bei komponentenweise partitionierten Transitionsrelationen möglichst viele Ähnlichkeiten zwischen Partitionen für verschiedene Komponenten ausnutzen zu können. Wie in Abschnitt 2.3 beschrieben, können Zustände und Transitionen eines asynchronen nebenläufigen Systems aus m-Komponenten mit charakteristischen Funktionen über einer Menge $X = \{x_1, x_2, ..., x_{2 \cdot N}\}$ von tatsächlichen Booleschen Variablen repräsentiert werden. Dabei gilt $N = n_g + \sum_{i=1}^{m} n_{l_i}$, wobei n_g die benötigte Anzahl an Variablen zur Kodierung des globalen Zustands des Systems und n_{l_i} die zur Kodierung des lokalen Zustands einer Komponente i benötigte Anzahl an Variablen ist.

Ein TLEBDD zur Repräsentation von Transitionen einer Komponente i besteht aus einem BDD über einer reduzierten Variablenmenge $Y = \{y_1, y_2, ..., y_{2 \cdot (n_g + n_{l_i})}\}$ zur Kodierung der Zustandsänderungen des globalen Zustands und des lokalen Zustands der Komponente i, sowie einer Zuordnungsliste (siehe Abschnitt 4.1). Die Zuordnungsliste gibt für jede reduzierte Variable die entsprechende tatsächliche Variable, die zur Kodierung des globalen Zustands oder des lokalen Zustands der Komponente i über den tatsächlichen Variablen benutzt wird, an.

ETLEBDDs sind eine Erweiterung von TLEBDDs zur gemeinsamen Speicherung von Transitionen mehrerer (≥ 2) Komponenten. Ein ETLEBDD besitzt wie ein TLEBDD einen BDD über einer reduzierten Variablenmenge. ETLEBDDs unterscheiden sich von TLEBDDs dadurch, dass sie zusätzlich anstatt einer Zuordnungsliste eine Zuordnungsliste für jede Komponente, für die Transitionen in einem ETLEBDD gespeichert sind, besitzen. Damit können die reduzierten Variablen des BDDs des ETLEBDDs für jede Komponente den entsprechenden tatsächlichen Variablen zugeordnet werden. Da zur Repräsentation aller in einem ETLEBDD gespeicherten Transitionen verschiedener Komponenten der gleiche BDD des ETLEBDDs verwendet wird, müssen bestimmte Voraussetzungen für eine Menge von gemeinsam in einem ETLEBDD zu repräsentierenden Transitionen gelten. Eine Menge von Transitionen verschiedener Komponenten, die gemeinsam in einem ETLEBDD repräsentiert werden kann, wird in Definition 5.1 als Menge isomorpher Transitionen eingeführt.

Definition 5.1. (Isomorphe Transitionen) *Gegeben sei ein asynchrones nebenläufiges System mit Transitionslokalität aus $m \geq 2$ Komponenten, eine Menge M aus k Komponenten des Systems ($2 \leq k \leq m$) und eine Kodierung von Zuständen und Transitionen des Systems wie in Abschnitt 2.3 angegeben. Seien außerdem eine Variablenanzahl $n_{l_{iso}}$ und eine Menge von reduzierten Variablen $Y = \{y_1, y_2, ..., y_{2 \cdot (n_g + n_{l_{iso}})}\}$ gegeben. Dann heißt eine Menge $R_T = \bigcup_{i \in M} R_{T_i}$, mit $R_{T_i} \subseteq R_i$, von Transitionen eine Menge isomorpher Transitionen, wenn gilt*

- $\forall i \in M$: $n_{l_i} = n_{l_{iso}}$ *und*

- *die Relation $R_{i_{PT}}$ mit $R_{i_{PT}} = \{((\vec{g}, l_i), (\vec{g}', l_i'))|$ die Zustandsänderung von $(\vec{g}, l_i)$ auf $(\vec{g}', l_i')$ ist Teil einer Transition in $R_{T_i}\}$ kann für alle k Komponenten mit dem gleichen BDD über den reduzierten Variablen der Menge Y repräsentiert werden.*

Für die in einer Menge von isomorphen Transitionen enthaltenen Transitionen können die durch Ausführen einer Transition resultierenden Änderungen des globalen Zustands und die des lokalen Zustands der ausführenden Komponente gemeinsam mit dem BDD eines ETLEBDDs repräsentiert werden. Sind die in Definition 5.1 angegebenen Eigenschaften erfüllt, heißt dies aber nur, dass diese Menge an Transitionen prinzipiell gemeinsam mit einem ETLEBDD repräsentiert werden kann. Bei einer konkreten Repräsentation durch einen ETLEBDD muss berücksichtigt werden, dass für die Repräsentation der Änderungen des lokalen Zustands der Komponenten im BDD des ETLEBDDs auch eine Zuordnung auf die reduzierten Variablen verwendet wird, die für die Transitionen aller Komponenten zu isomorphen BDDs führt. Dazu können für eine Menge isomorpher Transitionen mehrere Möglichkeiten existieren. Die zu einer verwendeten Kodierung über den reduzierten Variablen zu benutzende Zuordnung auf entsprechende tatsächliche Variablen, wird bei ETLEBDDs für jede Komponente in einer eigenen Zuordnungsliste gespeichert. Reduzierte Variablen zur Kodierung des globalen Zustands des Systems werden für alle Komponenten auf die gleichen tatsächlichen Variablen abgebildet. Zusätzlich werden in der vorliegenden Arbeit globale Variablen bei der Verwendung eines ETLEBDDs immer über den gleichen reduzierten Variablen kodiert. Die tatsächlichen Variablen zur Kodierung des lokalen Zustands zweier unterschiedlicher Komponenten sind aber verschieden. Daher unterscheiden sich für zwei verschiedene Komponenten die Mengen der in Zuordnungslisten für diese zur Kodierung des lokalen Zustands verwendeten tatsächlichen Variablen. Zur Speicherung der Zuordnungsinformationen in einem ETLEBDD, der eine Menge von zwischen k-Komponenten isomorphen Transitionen repräsentiert, besitzt ein ETLEBDD eine *k-fache Zuordnungsliste*. In der für diese nachfolgend angegebenen Definition 5.2 wird die in Abschnitt 4.1 zur Definition von TLEBDDs eingeführte Zuordnungsfunktion π_i benutzt, die für eine Komponente i eine Zuordnung der reduzierten Variablen auf die entsprechenden tatsächlichen Variablen angibt.

Definition 5.2. (k-fache Zuordnungsliste) *Gegeben sei ein asynchrones nebenläufiges System mit Transitionslokalität, eine Menge von k Komponenten des Systems und für jede Komponente $i \in \{1, ..., k\}$ eine Zuordnungsfunktion $\pi_i : \{1, 2, .., 2 \cdot (n_g + n_{l_i})\} \rightarrow \{1, 2, .., 2 \cdot N\}$. Dann ist eine k-fache Zuordnungsliste ein k-dimensionales Array, in dem für jede Komponente $i \in \{1, ..., k\}$ deren Zuordnungsliste gespeichert ist. Die Zuordnungsliste einer Komponente i, die für die Komponente die Abbildung der reduzierten Variablen auf die entsprechenden tatsächlichen Variablen angibt, ist dabei durch $[x_{\pi_i(1)}, ..., x_{\pi_i(2 \cdot (n_g + n_{l_i}))}]$ definiert.*

Durch die Verwendung einer k-fachen Zuordnungsliste können in einem ETLEBDD die Informationen zur Zuordnung der reduzierten Variablen auf die entsprechenden tatsächlichen Variablen für k Komponenten gespeichert werden. Damit können ETLEBDDs definiert werden.

Definition 5.3. (Erweitertes Transitionslokalitätsausnutzendes binäres Entscheidungsdiagramm (engl. extended transition locality exploiting BDD, ETLEBDD)) *Gegeben sei ein asynchrones nebenläufiges System mit Transitionslokalität aus $m > 1$ Komponenten und eine binäre Kodierung von Zuständen und Transitionen des Systems über Variablen der Menge $X = \{x_1, x_2, ..., x_{2 \cdot N}\}$, mit $N = n_g + \sum_{i=1}^{m} n_{l_i}$, wie in Abschnitt 2.3 angegeben. Dann ist ein ETLEBDD zur Repräsentation von Transitionen einer Menge von $k \geq 2$ Komponenten eines solchen Systems ein Tupel (G, l) mit folgenden Eigenschaften:*

- *G ist ein BDD über der Variablenmenge $Y = \{y_1, y_2, ..., y_{2 \cdot (n_g + n_{l_{iso}})}\}$, mit dem für eine Menge von zwischen k-Komponenten isomorphen Transitionen die dazu korrespondierenden Relationen $R_{i_{PT}}$ der k Komponenten repräsentiert werden. Die Variablen der Menge Y werden reduzierte Variablen genannt.*
- *l ist eine k-fache Zuordnungsliste, die benutzt wird um für jede Komponente $i \in \{1, ..., k\}$ die Menge der reduzierten Variablen $Y = \{y_1, y_2, ..., y_{2 \cdot (n_g + n_{l_{iso}})}\}$ auf die entsprechenden tatsächlichen Variablen aus X abzubilden. Eine Zuordnungsliste des Arrays für eine Komponente i besitzt dabei für jede reduzierte Variable y_q mit $q \in \{1, 2, ..., 2 \cdot (n_g + n_{l_{iso}})\}$ des BDDs G die entsprechende tatsächliche Variable $x_{\pi_i(q)}$. Dabei gilt:*
  - *$\forall i \in \{1, ..., k\}$ und $q_1, q_2 \in \{1, 2, ..., 2 \cdot (n_g + n_{l_{iso}})\}$ mit $q_1 \neq q_2$, dass $x_{\pi_i(q_1)} \neq x_{\pi_i(q_2)}$.*
  - *$\forall i, j \in \{1, ..., k\}$ mit $i \neq j$, $q_1, q_2 \in \{1, 2, ..., 2 \cdot (n_g + n_{l_{iso}})\}$ und tatsächliche Variablen $x_{\pi_i(q_1)}$, $x_{\pi_j(q_2)}$ zur Kodierung von lokalen Zuständen ist $x_{\pi_i(q_1)} \neq x_{\pi_j(q_2)}$.*
- *Bei Interpretation des BDDs G für eine Komponente i werden für tatsächliche Variablen zur Kodierung der Änderungen der lokalen Zustände der anderen $m - 1$ im System vorhandenen Komponenten implizit Identitätstransformationen angenommen.*

Durch die Verwendung einer geordneten k-fachen Zuordnungsliste werden ETLEBDDs kanonisch.

Definition 5.4. (Geordnete k-fache Zuordnungsliste) *Gegeben sei eine totale Ordnung $\prec$ über den tatsächlichen Variablen X. Eine k-fache Zuordnungsliste ist geordnet, wenn*

- *für die Zuordnungsliste jeder Komponente $z \in \{1, .., k\}$ im Array $x_{\pi_z(q)} \prec x_{\pi_z(q+1)}$ $\forall q \in \{1, .., n-1\}$ gilt und*
- *die Zuordnungslisten im Array so angeordnet sind, dass für zwei beliebige Arrayindizes $v, p \in \{1, .., k\}$, mit $v < p$ und die entsprechenden Komponenten K_v und K_p bei Definition von q wie oben angegeben gilt: $x_{\pi_{K_v}(q)} \preceq x_{\pi_{K_p}(q)}$.*

Die Kanonizität von ETLEBDDs wird nachfolgend in Satz 5.5 bewiesen.

Satz 5.5. (Kanonizität von ETLEBDDs) *Seien (G, b_g) und (H, b_h) zwei ETLEBDDs mit geordneten k-fachen Zuordnungslisten, die gemäß einer totalen Ordnung $\prec$ über den tatsächlichen Variablen X geordnet sind, $<_{level}$ die Variablenordnung der BDDs G und H der ETLEBDDs und g, h Boolesche Funktionen zur Repräsentation einer Menge von zwischen k Komponenten eines asynchronen nebenläufigen Systems mit Transitionslokalität isomorphen Transitionen. Wenn g durch (G, b_g) und h durch (H, b_h) repräsentiert wird, gilt $g = h$ genau dann, wenn $G = H$ und $b_g = b_h$ gilt.*

Beweis. Sei $b_g = b_h$, $G = H$ und K_i eine beliebige der k Komponenten und damit $K_i \in \{K_1, K_2, ..., K_k\}$. Dann ergibt sich durch Umbenennen der reduzierten Variablen der BDDs der ETLEBDDs gemäß der Zuordnungsliste der Komponente K_i aus b_g bzw. b_h und Einfügen der fehlenden Identitätstransformationen in die BDDs $g_{exp}[K_i] = \Delta(IdVar_i, G\{\vec{y} \leftarrow \vec{x}_{\pi_{K_i}}\})$ bzw. $h_{exp}[K_i] = \Delta(IdVar_i, H\{\vec{y} \leftarrow \vec{x}_{\pi_{K_i}}\})$, wobei Δ wie in Definition 4.4 in Abschnitt 4.1 angegeben definiert ist. $IdVar_i$ ist die Menge der Variablenpaare, für die für Komponente K_i implizit Identitätstransformationen zu berücksichtigen sind. Dies sind alle tatsächlichen Variablen außer den tatsächlichen Variablen, die in der Zuordnungsliste der Komponente K_i vorkommen. Daher ist die Menge $IdVar_i$ bei der Erzeugung von g_{exp} und h_{exp} gleich. Weil auch $G = H$ gilt und K_i eine beliebige der k-Komponenten ist, gilt $\forall K_i \in \{K_1, K_2, ..., K_k\}$ ist $g_{exp}[K_i] = h_{exp}[K_i]$ und damit auch $g = h$.
Nun sei $g = h$. Da die k-fachen Zuordnungslisten der ETLEBDDs geordnet sind und für jede Komponente K_i jeweils die gleichen tatsächlichen Variablen den reduzierten Variablen zugeordnet werden müssen, gibt es nur eine eindeutige geordnete k-fache Zuordnungsliste. Daher gilt $b_g = b_h$. Wäre bei gleicher Variablenordnung $<_{level}$ $G \neq H$, dann müssten die ETLEBDDs (G, b_g) bzw. (H, b_h) unterschiedliche Zuordnungslisten $b_g \neq b_h$ besitzen, damit $g = h$ gilt. Daher gilt auch $G = H$. □

Den BDD zu einer durch einen ETLEBDD repräsentierten Booleschen Funktion kann man durch separate Expansion des BDDs des ETLEBDDs für jede Komponente, für die im ETLEBDD Transitionen gespeichert sind, und die anschließende Vereinigung der resultierenden BDDs erhalten. Damit kann die durch einen ETLEBDD $E = (G, l)$ mit einer k-fachen Zuordnungsliste repräsentierte Boolesche Funktion über einer Menge X von tatsächlichen Variablen, wie folgt mit der in Abschnitt 4.1 eingeführten Operation Δ beschrieben werden:

$$E(\vec{x}) = \bigcup_{i \in \{1,2,...,k\}} \Delta(IdVar_i, G\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}). \tag{5.1}$$

Dabei ist $IdVar_i$ hier die Menge der Variablenpaare, für die für eine Komponente i implizit Identitätstransformationen zu berücksichtigen sind. Für eine Komponente i enthält die Menge $IdVar_i$ alle tatsächlichen Variablen außer den tatsächlichen Variablen, die in der Zuordnungsliste der Komponente i vorkommen. ETLEBDDs können mit beliebigen Variablenordnungen und auch mit Verfahren zum dynamischen Umordnen von Variablen verwendet werden. In Abschnitt 5.3 wird ein Algorithmus zur kombinierten Berechnung des relationalen Produkts zwischen einem ETLEBDD und einem BDD vorgestellt. Bei diesem ist keine vorherige Expansion des ETLEBDDs in einen entsprechenden BDD nötig.

Allerdings müssen vor Benutzung des Algorithmus die Variablenordnung der reduzierten Variablen des ETLEBDDs und die Variablenordnung der tatsächlichen Variablen eines mit diesem zu kombinierenden BDDs angeglichen werden. Dazu wird im nachfolgenden Abschnitt auf verträgliche ETLEBDD- und BDD-Variablenordnungen eingegangen.

5.2 Variablenordnungen von ETLEBDDs und BDDs

Für die direkte Kombination von TLEBDDs und BDDs wurden Voraussetzungen für die Variablenordnung der reduzierten Variablen des TLEBDDs und für die Variablenordnung der tatsächlichen Variablen des BDDs in Abschnitt 4.2 beschrieben. Der in Abschnitt 4.3 präsentierte Algorithmus zur kombinierten Berechnung des relationalen Produkts mit einem TLEBDD kann mit allen verschachtelten Variablenordnungen, die nach der in Abschnitt 4.2 angegebenen Definition konform sind, benutzt werden. Im Gegensatz zu einem TLEBDD, in dem nur Transitionen einer Komponente gespeichert werden, können mit einem ETLEBDD Transitionen verschiedener Komponenten gleichzeitig repräsentiert werden.

Der Algorithmus zur kombinierten Berechnung des relationalen Produkts mit einem ETLEBDD, der anschließend in Abschnitt 5.3 vorgestellt wird, erlaubt die simultane Berechnung des relationalen Produkts für alle Komponenten, für die in einem ETLEBDD Transitionen gespeichert sind. Bei der Definition der Konformität einer TLEBDD- und einer BDD-Variablenordnung in Abschnitt 4.2 wurden die für die reduzierten Variablen des TLEBDDs und die für die diesen entsprechenden tatsächlichen Variablen erlaubten Anordnungen in einer Variablenordnung beschränkt. Soll das relationale Produkt mit dem für TLEBDDs vorgestellten Algorithmus für eine Komponente berechnet werden, sind für die Anordnung der weiteren zur Kodierung des lokalen Zustands von anderen Komponenten verwendeten tatsächlichen Variablen, außer der Benutzung einer verschachtelten Variablenordnung, keine weiteren Beschränkungen nötig. Solche tatsächlichen Variablen werden in einem TLEBDD implizit als Variablen für Identitätstransformationen berücksichtigt und es sind in einem TLEBDD keine entsprechenden Knoten dafür vorhanden. Der Algorithmus zur kombinierten Berechnung des relationalen Produkts zwischen einem ETLEBDD und einem BDD aus Abschnitt 5.3 kann das relationale Produkt für alle Komponenten, für die Transitionen in einem ETLEBDD vorhanden sind, simultan berechnen. Damit dies möglich ist, muss die in Abschnitt 4.2 eingeführte Definition der Konformität einer TLEBDD- und einer BDD-Variablenordnung erweitert werden.

Definition 5.6. (Konformität einer ETLEBDD- und einer BDD-Variablenordnung) *Gegeben sei ein asynchrones nebenläufiges System mit Transitionslokalität aus m-Komponenten und ein ETLEBDD über einer Menge* $Y = \{y_1, y_2, ..., y_{2\cdot(n_g+n_{l_{iso}})}\}$ *von reduzierten Booleschen Variablen, der isomorphe Transitionen einer Menge* $M = \{K_1, K_2, ..., K_k\}$ *von* $2 \leq k \leq m$ *Komponenten beinhaltet. Seien außerdem ein BDD über einer Menge* $X = \{x_1, x_2, ..., x_{2\cdot N}\}$ *von tatsächlichen Booleschen Variablen, eine Variablenordnung der reduzierten Variablen des BDDs des ETLEBDDs* $<_{ETLEBDD}$ *und eine Variablenordnung der tatsächlichen Variablen des BDDs* $<_{BDD}$ *gegeben. Dann*

heißen die Variablenordnungen des ETLEBDDs $<_{ETLEBDD}$ und des BDDs $<_{BDD}$ konform, wenn gilt:

$$\forall z \in \{1, 2, ..., k\}, \forall y_i, y_j \in Y \; mit \; y_i \neq y_j : y_i <_{ETLEBDD} y_j \Leftrightarrow x_{\pi_z(i)} <_{BDD} x_{\pi_z(j)} \quad (5.2)$$

Eine ETLEBDD- und eine BDD-Variablenordnung sind damit konform, wenn für alle Komponenten, für die im ETLEBDD Transitionen gespeichert werden, die für die Konformität einer TLEBDD- und einer BDD-Variablenordnung (Definition 4.7) geforderte Bedingung erfüllt ist. Bei Verwendung von TLEBDDs können für alle Komponenten und TLEBDDs nach Definition 4.7 konforme und verschachtelte Variablenordnungen, für Bildberechnungen mit dem Algorithmus zur kombinierten Berechnung des relationalen Produkts mit einem TLEBDD mit allen TLEBDDs benutzt werden. Damit können Umordnungen der Variablenordnung zwischen Bildberechnungen mit TLEBDDs mit verschiedenen Anforderungen an die Variablenordnung vermieden werden. Bei konformen ETLEBDD- und BDD-Variablenordnungen kommen Knoten für korrespondierende reduzierte und tatsächliche Variablen auf allen Pfaden des ETLEBDDs und des BDDs in der gleichen Reihenfolge vor, wodurch diese bei der Berechnung des relationalen Produkts korrekt zusammen durchlaufen werden können. Für die Benutzung des im nächsten Abschnitt präsentierten Algorithmus muss es sich zusätzlich um verschachtelte Variablenordnungen handeln und die globalen Variablen eines Systems müssen für alle Komponenten, für die Transitionen im ETLEBDD gespeichert sind, über den gleichen reduzierten Variablen kodiert worden sein. Soll die Berechnung des relationalen Produkts mit einem ETLEBDD nur für eine Teilmenge der Komponenten erfolgen, für die im ETLEBDD Transitionen gespeichert sind, reicht für diese Bildberechnungen die Konformität der Variablenordnungen bezüglich dieser Teilmenge von Komponenten aus.

5.3 Algorithmus zur Berechnung des relationalen Produkts mit ETLEBDDs

In diesem Abschnitt wird für verschachtelte Variablenordnungen ein Algorithmus zur kombinierten Berechnung des relationalen Produkts für die Nachfolgerberechnung, zwischen einem ETLEBDD zur Repräsentation einer Menge von Transitionen und einem BDD zur Repräsentation einer zu explorierenden Zustandsmenge, vorgestellt. Mit einem ETLEBDD kann eine Transitionen mehrerer Komponenten beinhaltende Menge isomorpher Transitionen, gemeinsam unter Verwendung des gleichen BDDs des ETLEBDDs repräsentiert werden. Für verschiedene Komponenten unterscheidet sich lediglich die Zuordnung der reduzierten Variablen, über denen der BDD des ETLEBDDs definiert ist, auf die entsprechenden tatsächlichen Variablen. Dazu besitzt ein ETLEBDD für jede Komponente, für die Transitionen im ETLEBDD gespeichert werden, eine separate Zuordnungsliste. Wie auch der in Abschnitt 4.3 vorgestellte Algorithmus zur kombinierten Berechnung des relationalen Produkts mit einem TLEBDD, führt der hier vorgestellte Algorithmus eine kombinierte Berechnung der im relationalen Produkt enthaltenen Operationen durch.

Außerdem kann der Algorithmus das relationale Produkt für alle Komponenten, für die Transitionen im ETLEBDD gespeichert sind, auf einmal berechnen. Dazu ist keine vorherige Expansion des ETLEBDDs in einen entsprechenden BDD zur Repräsentation der im ETLEBDD enthaltenen Transitionen notwendig, wodurch die gewöhnlichen Algorithmen zur Berechnung des relationalen Produkts zwischen BDDs benutzt werden könnten. Stattdessen wird der ETLEBDD bei der Berechnung des relationalen Produkts direkt mit einem BDD zur Repräsentation einer Zustandsmenge verknüpft. Die Zuordnung der reduzierten Variablen des BDDs des ETLEBDDs zu den entsprechenden tatsächlichen Variablen wird für die einzelnen Komponenten erst während der Ausführung des Algorithmus durchgeführt. Dadurch können der Zeitaufwand zum Expandieren eines ETLEBDDs in einen entsprechenden BDD und der für den BDD benötigte Speicherbedarf eingespart werden. Der dazu hier vorgestellte Algorithmus führt die Kombination des ETLEBDDs mit dem BDD zur Repräsentation der zu explorierenden Zustandsmenge dabei beginnend mit den Wurzelknoten der beiden Datenstrukturen für mehrere Komponenten gemeinsam durch. Die gemeinsame Berechnung wird solange fortgeführt, wie für mehrere Komponenten die gleichen Berechnungen auszuführen sind. Für mehrere Komponenten gleich durchzuführende Berechnungen treten bei mit den Wurzelknoten beginnenden Berechnungspfaden auf, wenn die in den Zuordnungslisten angegebene Zuordnungsinformation zur Zuordnung auf die tatsächlichen Variablen für zu betrachtende Knoten des ETLEBDDs gleich ist. Bei Verwendung komponentenweise partitionierter Transitionsrelationen aus BDDs oder TLEBDDs müssten solche sich entsprechenden Teilberechnungen bei der Berechnung des relationalen Produkts für jede Partition der Transitionsrelation separat ausgeführt werden. Im Gegensatz dazu führt der hier vorgestellte Algorithmus diese nur einmal aus. Gleiche Teilberechnungen treten zum Beispiel für Knoten für globale Variablen, oder bei Knoten für Variablen, die für mehrere Komponenten Variablen zur Kodierung einer Identitätstransformation sind, auf. Für die Benutzbarkeit des Algorithmus müssen die globalen Variablen eines Systems für alle Komponenten, für die Bildberechnungen ausgeführt werden sollen, über den gleichen reduzierten Variablen des BDDs des ETLEBDDs kodiert werden. Dadurch werden reduzierte Variablen von Knoten für globale Variablen für alle Komponenten auf die gleichen tatsächlichen Variablen abgebildet. Die in dieser Arbeit in Abschnitt 8.5 und Abschnitt 8.6 präsentierten experimentellen Ergebnisse zeigen, dass die Benutzung des hier vorgestellten Algorithmus zu großen Laufzeitverringerungen und Einsparungen beim Speicherbedarf führen kann.

Für die Benutzbarkeit des im Folgenden vorgestellten Algorithmus müssen bestimmte Voraussetzungen für die Variablenordnungen der reduzierten Variablen des BDDs des ETLEBDDs und der tatsächlichen Variablen gelten. Diese müssen verschachtelte Variablenordnungen sein, die nach der in Abschnitt 5.2 angegebenen Definition konform sind. Bei Verwendung von Verfahren zum dynamischen Verändern der Variablenordnung muss darauf geachtet werden, dass die dadurch entstehenden Variablenordnungen diese Voraussetzungen vor der Ausführung des Algorithmus erfüllen. Der gewöhnliche BDD zur Repräsentation der in einem ETLEBDD gespeicherten Transitionen, kann durch Expansion des BDDs des ETLEBDDs für jede Komponente, für die Transitionen im ETLEBDD gespeichert sind, und anschließender Vereinigung der daraus resultierenden BDDs erzeugt

werden. Das relationale Produkt mit einem ETLEBDD mit Transitionen von k Komponenten kann unter Benutzung der in Abschnitt 4.1 eingeführten Operation Δ wie folgt beschrieben werden:

$$\exists\vec{x}[(\Delta(IdVar_1, R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_1}\}) \vee ... \vee \Delta(IdVar_k, R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_k}\})) \wedge S(\vec{x})]\{\vec{x}' \leftarrow \vec{x}\} \quad (5.3)$$

Dabei beinhalten die Mengen $IdVar_i$, wie die in Abschnitt 4.1 beschriebene Menge $IdVar$, korrespondierende Variablen für aktuelle Zustände und Nachfolgerzustände, für die für eine Komponente i Identitätstransformationen aufgrund der Transitionslokalität einzufügen sind. Dies sind für eine Komponente i die tatsächlichen Variablen, ohne die tatsächlichen Variablen zur Kodierung der Änderungen des globalen Zustands und der des lokalen Zustands der Komponente i. Da sich die tatsächlichen Variablen zur Kodierung des lokalen Zustands zweier verschiedener Komponenten eines asynchronen nebenläufigen Systems mit Transitionslokalität unterscheiden, gilt für zwei Komponenten i und j, mit $i \neq j$, auch $IdVar_i \neq IdVar_j$. Bei der Einführung von ETLEBDDs (siehe Definition 5.3) wurden wie bei der Einführung von TLEBDDs (siehe Definition 4.3) tatsächliche Variablen der Menge $X = \{x_1, x_2, ..., x_{2 \cdot N}\}$ verwendet. Im in Formel 5.3 angegebenen relationalen Produkt werden stattdessen aus Veranschaulichungsgründen zur Kodierung von tatsächlichen Variablen für aktuelle Zustände Variablen der Menge $X = \{x_1, x_2, ..., x_N\}$ und für tatsächliche Variablen für Nachfolgerzustände Variablen der Menge $X' = \{x'_1, x'_2, ..., x'_N\}$ verwendet. Die Repräsentation durch tatsächliche Variablen der Menge $X = \{x_1, x_2, ..., x_{2 \cdot N}\}$ kann daraus durch Indextransformation und Vereinigung von Variablen x_i mit den neuen Indizes erhalten werden.

Pseudocode zu den Hauptfunktionen des Algorithmus zur kombinierten Berechnung des relationalen Produkts mit einem ETLEBDD ist in den nachfolgend aufgeführten Algorithmen Algorithmus 14 (Funktion *RelProductETLEBDD()*), Algorithmus 15 (Funktion *StraightExecution()*) und Algorithmus 16 (Funktion *ExecutionSplit()*) angegeben. Von diesen wird bei Aufrufen des Algorithmus immer zuerst die Funktion *RelProductETLEBDD()* aufgerufen. In dieser finden Aufrufe der Funktionen *StraightExecution()* und *ExecutionSplit()* statt. Bei der Berechnung des relationalen Produkts geht der hier vorgestellte Algorithmus nach dem bei Operationen zwischen BDDs üblicherweise verwendeten rekursiven Berechnungsprinzip vor, bei dem in einem Rekursionsschritt eine Aufspaltung in Teilprobleme für die Belegung einer Booleschen Top-Variable mit den beiden Booleschen Werten erfolgt. Die rekursiv für die Teilprobleme ermittelten Ergebnisse werden anschließend zum Ergebnis des Rekursionsschritts vereinigt. Vor rekursiven Aufrufen des Algorithmus für Teilprobleme wird geprüft, ob im aktuellen Rekursionsschritt ein Terminalfall der rekursiven Berechnung vorliegt. Bei nicht vorliegendem Terminalfall wird als nächstes die Ergebnistabelle abgefragt und ermittelt, ob das Ergebnis der aktuell auszuführenden Berechnung bereits in der Ergebnistabelle gespeichert ist. Liegt kein Terminalfall vor und das Ergebnis der auszuführenden Berechnung konnte nicht in der Ergebnistabelle gefunden werden, wird die Berechnung durch rekursive Aufrufe des Algorithmus für Teilprobleme fortgesetzt. In der Funktion *RelProductETLEBDD()* wird dazu wenn weitere rekursive Aufrufe durchzuführen sind, zuerst die zur reduzierten Variablen des Wurzelknotens des BDDs des ETLEBDDs korrespondierende tatsächliche Variable *mappedVariable* ermittelt.

Für diese tatsächliche Variable ist der Wurzelknoten dieses BDDs im Rekursionsschritt zu berücksichtigen. Die Funktion *RelProductETLEBDD()* ruft falls weitere rekursive Aufrufe des Algorithmus durchgeführt werden, die in Algorithmus 15 angegebene Funktion *StraightExecution()* oder die in Algorithmus 16 angegebene Funktion *ExecutionSplit()* auf. In diesen werden dann die weiteren rekursiven Aufrufe des Algorithmus *RelProductETLEBDD()* durchgeführt.

Das Vorgehen des Algorithmus bei der rekursiven kombinierten Berechnung des relationalen Produkts kann direkt aus Formel 5.3 abgeleitet werden. Dies wird in Abschnitt 5.6 im Beweis zu Satz 5.7 durchgeführt, wodurch dort die Korrektheit dieses Vorgehens gezeigt wird. Im Algorithmus werden mit den Wurzelknoten der beteiligten Entscheidungsdiagramme beginnend für mehrere Komponenten gleich durchzuführende Teilberechnungen solange dies möglich ist, gemeinsam und nur einmal durchgeführt. Daher sind die in Satz 5.7 angegebenen Aufspaltungen von Berechnungen für k-Komponenten, für die in einem Rekursionsschritt noch Berechnungen auszuführen sind, angegeben. Die in den nachfolgend angegebenen Aufspaltungen verwendete Menge $M = \{1, 2, ..., k\}$ besteht aus den Indizes der k-Komponenten. Im Algorithmus *RelProductETLEBDD()* werden bei Rekursionsschritten mit einer tatsächlichen Variablen zur Kodierung des globalen Zustands eines Systems als Top-Variable υ und bei Rekursionsschritten mit einer tatsächlichen Variablen zur Kodierung des lokalen Zustands einer Komponente mit einem Index $i \in M$ als Top-Variable, unterschiedliche Aufspaltungen in Teilprobleme verwendet. Außerdem werden in beiden Fällen verschiedene Vorgehensweisen für eine Top-Variable für aktuelle Zustände ($\upsilon \in X$) und für eine Top-Variable für Nachfolgerzustände ($\upsilon \in X'$) benutzt. Für eine tatsächliche Variable υ zur Kodierung des globalen Zustands eines Systems ($\upsilon \in X$ oder $\upsilon \in X'$), wird im Algorithmus die folgende in Satz 5.7 angegebene Aufspaltung in Teilprobleme verwendet (für die Zuordnungsinformationen zu einer reduzierten Variablen y_p zur Kodierung des globalen Zustands gilt dabei $\forall i, j \in M : \pi_i(y_p) = \pi_j(y_p)$, weshalb in den folgenden beiden Fällen zur besseren Verständlichkeit für solche Variablen eine globale Zuordnung π_g verwendet wird):

- υ tatsächliche Variable für aktuelle Zustände ($\upsilon \in X$):

$$\exists \vec{x}[(\Delta(IdVar_1, R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_1}\}) \vee ... \vee \Delta(IdVar_k, R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_k}\})) \wedge S(\vec{x})]\{\vec{x}' \leftarrow \vec{x}\}$$
$$= (\bigvee_{i=1}^{k} (\exists \vec{x}(\Delta(IdVar_i, R(\vec{y})|_{y_{\pi_g^{-1}(\upsilon)}=1}\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}) \wedge S(\vec{x})|_{\upsilon=1}))\{\vec{x}' \leftarrow \vec{x}\}) \vee$$
$$(\bigvee_{i=1}^{k} (\exists \vec{x}(\Delta(IdVar_i, R(\vec{y})|_{y_{\pi_g^{-1}(\upsilon)}=0}\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}) \wedge S(\vec{x})|_{\upsilon=0}))\{\vec{x}' \leftarrow \vec{x}\})$$

Algorithmus 14: Kombinierte Berechnung des relationalen Produkts mit durch einen ETLEBDD repräsentierter Menge von Transitionen.

```
1  RelProductETLEBDD(ETLEBDD_BDD R, MappingList mapping, ROBDD S,
2                    Index compIndex, ExecutionMode mode, int numCompsSplit)
3  if IsTerminalCase(R, S, compIndex, mode) then
4  |  return TerminalCase(R, S, compIndex, mode);
5  else
6  |  if ComputedTableHasEntry(RelProdETLEBDD, R, S, compIndex,
7  |                            mode) then
8  |  |  return ComputedTable(RelProdETLEBDD, R, S, compIndex, mode);
9  |  else
10 |  |  if !IsTerminalVertex(R) then
11 |  |  |  mappedVariable = mapping[compIndex][VariableRoot(R)];
12 |  |  else
13 |  |  |  mappedVariable = TerminalValue;
14 |  |  υ = TopVariable(mappedVariable, VariableRoot(S));
15 |  |  if υ ==mappedVariable then
16 |  |  |  w = VariableRoot(R);
17 |  |  |  R_{w_1} = R|_{w=1};
18 |  |  |  R_{w_0} = R|_{w=0};
19 |  |  else
20 |  |  |  R_{w_1} = R_{w_0} = R;
21 |  |  if mode==iso || !IsLocalCompVar(υ) || (IsLocalCompVar(υ) &&
   |  |     previousSplitComp[CompLocVar(υ)]==TRUE) then
22 |  |  |  Result = StraightExecution(R, R_{w_1}, R_{w_0}, mapping, S, υ, compIndex,
23 |  |  |                             mode, numCompsSplit);
24 |  |  else
25 |  |  |  Result = ExecutionSplit(R, R_{w_1}, R_{w_0}, mapping, S, υ, compIndex,
26 |  |  |                          numCompsSplit);
27 |  |  InsertComputedTable(RelProdETLEBDD, R, S, compIndex, Result,
   |  |                      mode);
28 |  |  return Result;
```

Algorithmus 15: Aufspaltung der Berechnung eines Rekursionsschritts des Algorithmus *RelProductETLEBDD()* in Teilprobleme, wenn keine Aufsplittung der Berechnung in für Komponenten unterschiedliche Berechnungen erfolgt.

```
1  StraightExecution(ETLEBDD_BDD R, ETLEBDD_BDD R_w1, ETLEBDD_BDD R_w0,
2                    MappingList mapping, ROBDD S, Variable v,
3                    Index compIndex, ExecutionMode mode, int numCompsSplit)
4  if mode==sim && R != R_w1 && IsGlobalVarVertex(R) &&
5    IsLocalVarVertex(R_w1) then
6  |  newCompIndex=GetLowestComp(R_w1, mapping);
7  |  R_1 = RelProductETLEBDD(R_w1, mapping, S|_v=1, newCompIndex,
8  |                           mode, numCompsSplit);
9  else
10 |  R_1 = RelProductETLEBDD(R_w1, mapping, S|_v=1, compIndex, mode,
11 |                           numCompsSplit);
12 if mode==sim && R != R_w0 && IsGlobalVarVertex(R) &&
13   IsLocalVarVertex(R_w0) then
14 |  newCompIndex=GetLowestComp(R_w0, mapping);
15 |  R_0 = RelProductETLEBDD(R_w0, mapping, S|_v=0, newCompIndex,
16 |                           mode, numCompsSplit);
17 else
18 |  R_0 = RelProductETLEBDD(R_w0, mapping, S|_v=0, compIndex, mode,
19 |                           numCompsSplit);
20 Result = ResultCalculation(R_1, R_0, v, compIndex);
21 return Result;
```

Algorithmus 16: Funktion, die der Algorithmus *RelProductETLEBDD()* zur rekursiven Aufspaltung der Berechnung in Teilprobleme, bei sich für Komponenten unterscheidenden auszuführenden Teilberechnungen aufruft.

```
1  ExecutionSplit(ETLEBDD_BDD R, ETLEBDD_BDD R_w1, ETLEBDD_BDD R_w0,
2                 MappingList mapping, ROBDD S, Variable v, Index compIndex,
3                 int numCompsSplit)
4  compIso=CompLocVar(v);
5  IncrementRemovedCounter(v);
6  previousSplitComp[compIso]=TRUE;
7  if IsCurrentStateVar(v) then
8  |   R_1 = RelProductETLEBDD(R_w1, mapping, S|_v=1, compIso, iso,
9  |                            numCompsSplit);
10 |   R_0 = RelProductETLEBDD(R_w0, mapping, S|_v=0, compIso, iso,
11 |                            numCompsSplit);
12 else
13 |   R_1 = RelProductETLEBDD(R_w1, mapping, S, compIso, iso,
14 |                            numCompsSplit);
15 |   R_0 = RelProductETLEBDD(R_w0, mapping, S, compIso, iso,
16 |                             numCompsSplit);
17 Result_split1 = ResultCalculation(R_1, R_0, v, compIso);
18 if IsGlobalVarVertex(R) then
19 |   newCompIndex=GetNextLocVarComp(v);
20 else
21 |   newCompIndex=GetLowestComp(R, mapping);
22 numCompsSplit= numCompsSplit +1;
23 z = v;
24 if IsNextStateVar(v) then
25 |   z =CurrStateVar(v);
26 newMode = sim;
27 if numCompsSplit==numberOfComponents−1 then
28 |   newMode = iso;
29 R_1 = RelProductETLEBDD(R, mapping, S|_z=1, newCompIndex, newMode,
30                         numCompsSplit);
31 R_0 = RelProductETLEBDD(R, mapping, S|_z=0, newCompIndex, newMode,
32                         numCompsSplit);
33 Result_split2 = ResultCalculation(R_1, R_0, v, newCompIndex);
34 Result = OR(Result_split1, Result_split2);
35 numCompsSplit= numCompsSplit −1;
36 previousSplitComp[compIso]=FALSE;
37 DecrementRemovedCounter(v);
38 return Result;
```

- υ tatsächliche Variable für Nachfolgerzustände ($\upsilon \in X'$):

$$\begin{aligned}
&\exists\vec{x}[(\Delta(IdVar_1, R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_1}\}) \vee ... \vee \Delta(IdVar_k, R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_k}\})) \wedge S(\vec{x})]\{\vec{x}' \leftarrow \vec{x}\} \\
&=((\upsilon)\{\vec{x}' \leftarrow \vec{x}\} \cdot (\bigvee_{i=1}^{k}(\exists\vec{x}(\Delta(IdVar_i, R(\vec{y})|_{y_{\pi_g^{-1}(\upsilon)}=1}\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}) \wedge S(\vec{x})))\{\vec{x}' \leftarrow \vec{x}\}))\vee \\
&((\overline{\upsilon})\{\vec{x}' \leftarrow \vec{x}\} \cdot (\bigvee_{i=1}^{k}(\exists\vec{x}(\Delta(IdVar_i, R(\vec{y})|_{y_{\pi_g^{-1}(\upsilon)}=0}\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}) \wedge S(\vec{x})))\{\vec{x}' \leftarrow \vec{x}\}))
\end{aligned}$$

In beiden Fällen sind hier die in den Teilproblemen zu verwendenden Kofaktoren für alle Komponenten gleich und die Kombination der für die Teilprobleme ermittelten Ergebnisse erfolgt auch auf die gleiche Weise. Daher sind in einem solchen Rekursionsschritt für alle Komponenten, für die noch Berechnungen auszuführen sind, die gleichen Berechnungen durchzuführen. Der Algorithmus *RelProductETLEBDD()* führt hier zwei rekursive Aufrufe durch, einen für die Belegung der Top-Variablen eines solchen Rekursionsschritts mit jedem der zwei Booleschen Werte. In beiden rekursiven Aufrufe werden dabei jeweils die entsprechenden Teilprobleme für alle noch zu betrachtenden Komponenten berücksichtigt. Nachdem die Ergebnisse der Teilprobleme ermittelt wurden, werden diese zum Ergebnis des Rekursionsschritts kombiniert. Die hier beschriebenen rekursiven Aufrufe und die Kombination der Teilergebnisse werden im Algorithmus von der in Algorithmus 15 angegebenen Funktion *StraightExecution()* ausgeführt, die dazu von der Hauptfunktion *RelProductETLEBDD()* aufgerufen wird (Zeile 22 in Algorithmus 14). Ist die tatsächliche Variable υ eine Variable zur Kodierung von lokalen Zuständen einer Komponente mit einem Index $i \in M$, werden im Algorithmus die Folgenden in Satz 5.7 angegebenen Aufspaltungen in Teilprobleme benutzt:

- υ tatsächliche Variable für aktuelle Zustände ($\upsilon \in X$):

$$\begin{aligned}
&\exists\vec{x}[(\Delta(IdVar_1, R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_1}\}) \vee ... \vee \Delta(IdVar_k, R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_k}\})) \wedge S(\vec{x})]\{\vec{x}' \leftarrow \vec{x}\} \\
&=(\bigvee_{j\in M\setminus\{i\}} \upsilon \cdot (\exists\vec{x}(\Delta(IdVar'_j, R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_j}\}) \wedge S(\vec{x})|_{\upsilon=1}))\{\vec{x}' \leftarrow \vec{x}\})\vee \\
&(\bigvee_{j\in M\setminus\{i\}} \overline{\upsilon} \cdot (\exists\vec{x}(\Delta(IdVar'_j, R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_j}\}) \wedge S(\vec{x})|_{\upsilon=0}))\{\vec{x}' \leftarrow \vec{x}\})\vee \\
&((\exists\vec{x}(\Delta(IdVar, R(\vec{y})|_{y_{\pi_i^{-1}(\upsilon)}=1}\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}) \wedge S(\vec{x})|_{\upsilon=1}))\{\vec{x}' \leftarrow \vec{x}\})\vee \\
&((\exists\vec{x}(\Delta(IdVar, R(\vec{y})|_{y_{\pi_i^{-1}(\upsilon)}=0}\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}) \wedge S(\vec{x})|_{\upsilon=0}))\{\vec{x}' \leftarrow \vec{x}\})
\end{aligned}$$

- υ tatsächliche Variable für Nachfolgerzustände ($\upsilon \in X'$) und υ' korrespondierende tatsächliche Variable für aktuelle Zustände ($\upsilon \in X$):

$$\begin{aligned}
&\exists\vec{x}[(\Delta(IdVar_1, R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_1}\}) \vee ... \vee \Delta(IdVar_k, R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_k}\})) \wedge S(\vec{x})]\{\vec{x}' \leftarrow \vec{x}\} \\
&=(\bigvee_{j\in M\setminus\{i\}} \upsilon' \cdot (\exists\vec{x}(\Delta(IdVar'_j, R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_j}\}) \wedge S(\vec{x})|_{\upsilon'=1}))\{\vec{x}' \leftarrow \vec{x}\}) \vee \\
&(\bigvee_{j\in M\setminus\{i\}} \overline{\upsilon'} \cdot (\exists\vec{x}(\Delta(IdVar'_j, R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_j}\}) \wedge S(\vec{x})|_{\upsilon'=0}))\{\vec{x}' \leftarrow \vec{x}\}) \vee \\
&((\upsilon)\{\vec{x}' \leftarrow \vec{x}\} \cdot ((\exists\vec{x}(\Delta(IdVar, R(\vec{y})|_{y_{\pi_i^{-1}(\upsilon)}=1}\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}) \wedge S(\vec{x}))) \\
&\{\vec{x}' \leftarrow \vec{x}\})) \vee ((\overline{\upsilon})\{\vec{x}' \leftarrow \vec{x}\} \cdot ((\exists\vec{x}(\Delta(IdVar, R(\vec{y})|_{y_{\pi_i^{-1}(\upsilon)}=0}\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}) \wedge \\
&S(\vec{x})))\{\vec{x}' \leftarrow \vec{x}\}))
\end{aligned}$$

Bei Aufspaltung der Berechnung bezüglich einer tatsächlichen Variablen υ zur Kodierung von lokalen Zuständen einer Komponente mit einem Index $i \in M$, sind die Mengen $IdVar_j$ für Komponenten mit Indizes $j \in M$ mit $j \neq i$ nicht leer. Für diese Komponenten sind die Mengen $IdVar'_j$ hier in den Aufspaltungen jeweils die Mengen $IdVar_j$ ohne das Variablenpaar υ, υ', das für Komponenten mit Indizes $j \in M$ und $j \neq i$ ein Variablenpaar zur Kodierung eines Identitätsmusters ist. Es gilt $IdVar'_j = IdVar_j - \{(\upsilon,\ \upsilon')\}$ (bzw. $IdVar'_j = IdVar_j - \{(\upsilon',\ \upsilon)\}$, wenn υ' Variable für aktuelle Zustände und υ korrespondierende Variable für Nachfolgerzustände ist}). In beiden Fällen, für eine tatsächliche Variable für aktuelle Zustände und für eine tatsächliche Variable für Nachfolgerzustände, sind hier jeweils verschiedene Berechnungen für die Komponente mit dem Index i und für alle anderen zu betrachtenden Komponenten auszuführen. Die für alle anderen Komponenten im Rekursionsschritt auszuführenden Berechnungen sind dabei für all diese Komponenten, genauso wie die bei den Teilproblemen zu verwendenden Kofaktoren, gleich. Daher führt der Algorithmus in einem Rekursionsschritt mit für Komponenten unterschiedlich auszuführenden Berechnungen, für die Teilprobleme für eine Belegung der Top-Variable mit jedem der beiden Booleschen Werte jeweils einen rekursiven Aufruf für die Komponente mit dem Index i und ebenso einen rekursiven Aufruf für die Teilprobleme aller anderen Komponenten durch. Die dafür notwendigen Berechnungen werden im Algorithmus in der in Algorithmus 16 angegebenen Funktion *ExecutionSplit()* durchgeführt, die dazu in der Funktion *RelProductETLEBDD()* aufgerufen wird (Zeile 25 in Algorithmus 14). Die für eine Top-Variable zur Kodierung von lokalen Zuständen einer Komponente angegebenen Aufspaltungen in Teilprobleme, werden bei für alle Komponenten in einem Rekursionsschritt mit einer Top-Variable für lokale Zustände gleich durchzuführenden Berechnungen aber auch von der Funktion *StraightExecution()* verwendet. In diesem Fall wird von diesen Aufspaltungen in Teilprobleme nur die Aufspaltung, die für alle zu betrachtenden Komponenten durchzuführen und gültig ist, ausgeführt. Dieser Fall tritt zum Beispiel ein, wenn die Top-Variable eine Variable zur Kodierung des lokalen Zustands einer Komponente ist, für die im aktuellen Rekursionsschritt keine Berechnungen mehr durchzuführen sind. Die Top-Variable ist dann für alle im Rekursionsschritt zu betrachtenden Komponenten eine Variable zur Kodierung einer Identitätstransformation und die Funktion *RelProductET-*

LEBDD() ruft zur Durchführung der beiden weiteren rekursiven Aufrufe die Funktion *StraightExecution()* auf (Zeile 22 in Algorithmus 14).

Der Algorithmus *RelProductETLEBDD()* besitzt zwei Ausführungsmodi, den Modus *iso* und den Modus *sim*. Im Ausführungsmodus *iso* sind die vom Algorithmus ausgeführten Berechnungen nur für eine Komponente gültig. Wurde der Algorithmus im Ausführungsmodus *iso* aufgerufen, wird dieser Ausführungsmodus in den vom entsprechenden Rekursionsschritt ausgehenden nachfolgenden rekursiven Aufrufen beibehalten. Im Ausführungsmodus *iso* des Algorithmus wird zur Ausführung weiterer rekursiver Aufrufe des Algorithmus *RelProductETLEBDD()* immer die Funktion *StraightExecution()* aufgerufen. Die in diesem Fall in dieser durchgeführten Berechnungen entsprechen dem Vorgehen des in Abschnitt 4.3 vorgestellten Algorithmus zu kombinierten Berechnung des relationalen Produkts mit einem TLEBDD. Im Ausführungsmodus *sim* kann der Algorithmus hingegen für mehrere Komponenten gleich durchzuführende Berechnungen gemeinsam ausführen, wodurch die Berechnungen nur einmal erfolgen müssen. Ist der Algorithmus in seinem Ausführungsmodus *sim*, wird in der Funktion *RelProductETLEBDD()* die Funktion *StraightExecution()* oder die Funktion *ExecutionSplit()* zur Durchführung weiterer rekursiver Aufrufe aufgerufen. In Rekursionsschritten mit für die zu betrachtenden Komponenten unterschiedlich auszuführenden Berechnungen wird dabei die Funktion *ExecutionSplit()* aufgerufen, ansonsten die Funktion *StraightExecution()*. Von der Funktion *ExecutionSplit()* werden die weiteren rekursiven Aufrufe des Algorithmus für Berechnungen für die Komponente, für die durch die rekursiven Aufrufe alleine gültige Berechnungen durchgeführt werden sollen, im Ausführungsmodus *iso* des Algorithmus ausgeführt.

Im weiteren Verlauf dieses Abschnitts wird der Algorithmus *RelProductETLEBDD()* genauer beschrieben. Er besitzt folgende Übergabeparameter:

- den BDD R eines ETLEBDDs, der eine Menge von Transitionen repräsentiert,
- die Zuordnungsliste *mapping* dieses ETLEBDDs, die die Zuordnungslisten der einzelnen Komponenten beinhaltet,
- einen BDD S, mit dem die zu explorierende Zustandsmenge repräsentiert wird,
- den Komponentenindex *compIndex* der Komponente, deren Zuordnungsliste vom Algorithmus gerade zur Abbildung der reduzierten Variablen des BDDs des ETLEBDDs auf die entsprechenden tatsächlichen Variablen verwendet wird,
- den aktuellen Ausführungsmodus *mode* des Algorithmus und
- die Anzahl der Komponenten *numCompsSplit*, für die die Berechnung auf einem Berechnungspfad vom Algorithmus bereits in nur noch für eine Komponente gültige Berechnungen aufgespalten wurde.

Im Folgenden ist außerdem *numberOfComponents* die Anzahl der Komponenten, für die Transitionen in einem ETLEBDD gespeichert sind (*numberOfComponents*$= k$). Für diese Komponenten sind daher auch Informationen zur Zuordnung der reduzierten Variablen auf die entsprechenden tatsächlichen Variablen in der Zuordnungsliste vorhanden.

Der nachfolgend vorgestellte Algorithmus berechnet das relationale Produkt mit den in einem ETLEBDD für alle *numberOfComponents* Komponenten gespeicherten Transitionen. Eine Berechnung des relationalen Produkts mit Transitionen einer Teilmenge dieser Komponenten kann zum Beispiel durch eine Anpassung der im 1. Aufruf des Algorithmus verwendeten Übergabeparameter erreicht werden. Dazu ist die Zuordnungsliste auf Zuordnungsinformationen für die gewünschte Teilmenge der Komponenten anzupassen. Ebenfalls muss die Komponentenanzahl *numberOfComponents*, die die Anzahl der Komponenten für die das relationale Produkt zu berechnen ist angibt, angepasst werden. Eine weitere Möglichkeit der Beschränkung der Berechnung des relationalen Produkts auf eine Teilmenge der Komponenten wäre die Modifikation von Algorithmus 19, der in Abschnitt 5.4 zur Auswahl der nächsten Komponente deren Zuordnungsinformationen benutzt werden soll, vorgestellt wird. Nachfolgend wird die gemeinsame Berechnung des relationalen Produkts für alle Komponenten, für die in einem ETLEBDD Transitionen gespeichert sind, beschrieben.

Wie oben bereits aufgeführt, besitzt der Algorithmus *RelProductETLEBDD()* (Algorithmus 14) zwei Ausführungsmodi, den Modus *iso* und den Modus *sim*. Im Ausführungsmodus *iso* sind die vom Algorithmus ausgeführten Berechnungen nur für die Komponente gültig, auf deren Komponentenindex der Übergabeparameter *compIndex* gerade gesetzt ist. Im Ausführungsmodus *sim* kann der Algorithmus hingegen für mehrere Komponenten gleich durchzuführende Berechnungen gemeinsam ausführen. Sind für mehrere Komponenten Berechnungen durchzuführen, erfolgt der erste Aufruf des Algorithmus im Ausführungsmodus *sim* und die Anzahl der bereits erfolgten Aufspaltungen in nur noch für eine Komponente gültige Berechnungen *numCompsSplit* wird auf den Wert 0 gesetzt. Algorithmus 14 führt nur noch für eine Komponente gültige Berechnungen auf Berechnungspfaden in der Reihenfolge durch, in der die ersten für tatsächliche Variablen zur Kodierung von Änderungen des lokalen Zustands der Komponenten zu berücksichtigenden Knoten als Top-Variablen in Rekursionsschritten vorkommen. Dazu gibt der Übergabeparameter *compIndex* des Algorithmus den Index der Komponente an, deren Zuordnungsliste gerade zur Abbildung von reduzierten Variablen auf die zu diesen korrespondierenden tatsächlichen Variablen verwendet wird. Beim ersten Aufruf des Algorithmus muss diese Variable auf die korrekte Komponente gesetzt werden. Zur Ermittlung des Werts von *compIndex* wird dazu der Wurzelknoten des BDDs R des ETLEBDDs betrachtet. Ist dieser ein Knoten für eine tatsächliche Variable zur Kodierung von Änderungen des globalen Zustands des Systems, ist die Zuordnung der reduzierten Variable des Knotens zur korrespondierenden tatsächlichen Variablen für alle Komponenten gleich. Damit kann *compIndex* auf einen beliebigen Index für eine der *numberOfComponents* Komponenten, für die die Berechnung des relationalen Produkts durchgeführt wird, gesetzt werden. In diesem Abschnitt wird für den Index einer Komponente immer die Komponentennummer verwendet. Im Algorithmus wird *compIndex* zu Beginn in diesem Fall auf die Komponente mit dem niedrigsten Index unter den Komponenten, für die Berechnungen durchzuführen sind, gesetzt. Falls der Wurzelknoten des BDDs R des ETLEBDDs ein Knoten für eine Variable zur Kodierung des Zustandsübergangs des lokalen Zustands einer Komponente ist, wird *compIndex* auf die noch zu betrachtende Komponente mit der als erstes in der Variablenordnung noch

vorkommenden korrespondierenden tatsächlichen Variablen zur reduzierten Variablen des Wurzelknotens des BDDs R gesetzt. Dadurch wird im Algorithmus die Verknüpfung von Knoten des BDDs zur Repräsentation der zu explorierenden Zustandsmenge mit Knoten des BDDs R, in der Reihenfolge des Auftretens von Knoten für tatsächliche Variablen für lokale Zustände von Komponenten in der Variablenordnung möglich. Der Komponentenindex auf den *compIndex* dazu beim ersten Aufruf des Algorithmus zu setzen ist, kann durch Betrachten der Einträge zur reduzierten Variablen des Wurzelknotens des BDDs R in der Zuordnungsliste und der aktuellen Variablenordnung der tatsächlichen Variablen ermittelt werden. Ein Ansatz, der dazu im in diesem Abschnitt vorgestellten Algorithmus verwendet wird, ist in Abschnitt 5.4 angegeben. Dieser berücksichtigt auch auf einem Berechnungspfad, möglicherweise durch vorheriges Auftreten von Knoten für lokale Variablen, bereits erfolgte Aufspaltungen in nur noch für einzelne Komponenten gültige Berechnungen.

Algorithmus 17: Funktion *TerminalCase()*, die bei einem in Algorithmus 14 aufgetretenen Terminalfall den BDD mit dem Ergebnis des Terminalfalls berechnet und zurückgibt.

```
1  TerminalCase(ETLEBDD_BDD R, ROBDD S, Index compIndex,
2               ExecutionMode mode)
3  if IsTerminalVertexZero(R) || IsTerminalVertexZero(S) then
4      return BDDTerminalVertexZero;
5  if IsTerminalVertexOne(R) then
6      if IsTerminalVertexOne(S) || OnlyGlobalOrLocalVariablesFollowCompIndex(
7          VariableRoot(S), compIndex) then
8          return BDDTerminalVertexOne;
9      if OnlyIdentityVariablesFollowCompIndex(VariableRoot(S), compIndex)
10         && mode == iso then
11         return S;
```

In einem Rekursionsschritt prüft der Algorithmus *RelProductETLEBDD()* zuerst, ob ein Terminalfall der rekursiven Berechnung vorliegt. Bei aufgetretenem Terminalfall wird das Resultat des Terminalfalls durch die Funktion *TerminalCase()*, die in Algorithmus 17 angegeben ist, ermittelt und zurückgegeben (siehe Zeile 4 in Algorithmus 14). Die Korrektheit der verwendeten Terminalfälle und eine weitere Beschreibung dieser werden in Abschnitt 5.7 behandelt. Sie sind Anpassungen der in Abschnitt 4.3 für den Algorithmus für TLEBDDs präsentierten und beschriebenen Terminalfälle. Sind in einem Rekursionsschritt noch Berechnungen für mehrere Komponenten auszuführen, müssen bei Terminalfällen hier die Rückgabewerte all dieser Berechnungen berücksichtigt werden. In Algorithmus 17 liegen Terminalfälle mit dem Terminalknoten für den Booleschen Wert 0 als Rückgabewert vor, wenn der BDD R des ETLEBDDs oder der BDD S dem Terminalknoten für den Wert 0 entsprechen. Alle noch durchzuführenden Berechnungen besitzen dann den Terminalknoten für den Wert 0 als Ergebnis. Weitere Terminalfälle sind vorhan-

den, wenn der BDD R in einem Rekursionsschritt ein Terminalknoten für den Booleschen Wert 1 ist. Einer dieser Terminalfälle liegt vor, wenn zusätzlich der Wurzelknoten des BDDs S ein Terminalknoten für den Booleschen Wert 1 ist. Ein weiterer Terminalfall ist vorhanden, falls in der für die tatsächlichen Variablen verwendeten Variablenordnung für die Komponente, auf die *compIndex* gerade zeigt, ab der Variablen des Wurzelknotens des BDDs S nur noch Variablen zur Kodierung des globalen Zustands des Systems und des lokalen Zustands dieser Komponente vorkommen. Beide zuvor beschriebenen Terminalfälle sind sowohl im Ausführungsmodus *iso*, als auch im Ausführungsmodus *sim* des Algorithmus gültig (siehe Zeile 6 und 7 in Algorithmus 17) und liefern den Terminalknoten für den Booleschen Wert 1 als Rückgabewert. Der letzte Terminalfall gilt nur im Ausführungsmodus *iso* des Algorithmus und liefert den BDD S als Rückgabewert. Er liegt ebenfalls vor, wenn der Wurzelknoten des BDDs R des ETLEBDDs ein Terminalknoten für den Booleschen Wert 1 ist. Zusätzlich müssen in der Variablenordnung der tatsächlichen Variablen für die Komponente auf die *compIndex* gerade zeigt, ab der Variablen des Wurzelknotens des BDDs S nur noch Variablen zur Kodierung von Änderungen des lokalen Zustands anderer Komponenten vorkommen. Diese sind für die durch *compIndex* angegebene Komponente, für die damit im Ausführungsmodus *iso* gerade ausschließlich Berechnungen durchgeführt werden, Variablen zur Kodierung von Identitätstransformationen. Wie bereits beim relationalen Produkt mit einem TLEBDD erwähnt (siehe Abschnitt 4.3) kann die Entscheidung, dass für eine Komponente nur noch Variablen mit einer Interpretation vorkommen, schnell durchgeführt werden, wenn entsprechende Positionen in einer Variablenordnung für jede Komponente einmal ermittelt und abgespeichert werden. Eine Aktualisierung der gespeicherten Werte ist nur bei einer Veränderung der verwendeten Variablenordnung notwendig. In Algorithmus 17 erfolgt die Abfrage ob für eine Komponente in der Variablenordnung nur noch Variablen zur Kodierung des globalen Zustands oder des lokalen Zustands vorkommen, durch die Funktion *OnlyGlobalOrLocalVariablesFollowCompIndex()*. Das ausschließliche Vorkommen von Variablen für Identitätstransformationen, die Variablen zur Kodierung von Änderungen des lokalen Zustands anderer Komponenten sind, wird durch die Funktion *OnlyIdentityVariablesFollowCompIndex()* abgefragt.

Ist in einem Rekursionsschritt kein Terminalfall vorhanden, wird vom Algorithmus *RelProductETLEBDD()* geprüft, ob das Ergebnis der im aktuellen Rekursionsschritt durchzuführenden Berechnung bereits berechnet wurde und noch in der Ergebnistabelle gespeichert ist (siehe Zeilen 6 bis 8). Zur Sicherstellung der Eindeutigkeit von Zugriffen auf die Ergebnistabelle werden hier als Übergabeparameter neben der Operationsart (*RelProdETLEBDD*) und den Wurzelknoten der beteiligten BDDs, noch der Wert der Variable *compIndex* und der Ausführungsmodus des Algorithmus *mode* verwendet. Damit können nur Ergebnisse für den gleichen Ausführungsmodus aus der Ergebnistabelle zurückgegeben werden. Durch zusätzliche Verwendung des Übergabeparameters *compIndex*, müssen die zurückgegebenen Ergebnisse in beiden Ausführungsmodi auch für den gleichen übergebenen Komponentenindex berechnet worden sein.

Im Ausführungsmodus *sim* des Algorithmus können noch für mehrere Komponenten Berechnungen auszuführen sein. In den in der vorliegenden Arbeit verwendeten Verifikationsexperimenten und Variablenordnungen (sowohl ohne als auch mit dynamischem Verändern

der Variablenordnung) konnte der Wert von *compIndex* auch zur eindeutigen Kennzeichnung der Komponentenmenge, für die noch Berechnungen auszuführen waren, verwendet werden. Je nach Aussehen des BDDs R des ETLEBDDs und des BDDs S können sich theoretisch aber auch Situationen ergeben, in denen diese Übergabeparameter im Ausführungsmodus *sim* auch bei konformen und verschachtelten Variablenordnungen nicht mehr zur Sicherstellung der Eindeutigkeit von Zugriffen auf die Ergebnistabelle ausreichen. Ein Beispiel dafür wäre, wenn im BDD R für das erste in der Variablenordnung vorkommende korrespondierende Variablenpaar aus tatsächlichen Variablen zur Kodierung des lokalen Zustands von Komponenten auf einem Berechnungspfad kein entsprechender Knoten für eine reduzierte Variable vorhanden ist. Dann kann es sein, dass der entsprechende Teil des BDDs R einmal mit einem im BDD S für die Variablen vorhandenen Knoten und einmal ohne Knoten im BDD S betrachtet wird. Bei Auftreten eines Knotens im BDD S wird hier eine Aufspaltung der Berechnungen ausgeführt, wohingegen im anderen Fall die simultane Berechnung mit der gleichen Komponentenmenge fortgesetzt wird. In beiden Fällen kann im weiteren Verlauf der Berechnung ein Rekursionsschritt erreicht werden, in dem *compIndex* in beiden Fällen wieder auf den gleichen Wert zeigt. Es sind aber noch für unterschiedliche Komponentenmengen Berechnungen auszuführen. Bei den in dieser Arbeit mit ETLEBDDs verwendeten Variablenordnungen kommen die Booleschen Variablen zur Kodierung des lokalen Zustands von Komponenten immer aufeinanderfolgend vor (siehe Abschnitt 8.3). Daher kann der hier beschriebene Fall nur bei sehr unrealistischen Systemen eintreten, bei denen bei Verwendung von ETLEBDDs auf mindestens einem Pfad des ETLEBDDs keine Beschränkung für die Änderungen des lokalen Zustands vorhanden ist. Da der ETLEBDD für alle Komponenten verwendet wird, müsste dies für alle Komponenten gelten. Werden andere Variablenordnungen verwendet, können bei Zugriffen auf die Ergebnistabelle zur Behebung dieses Problems zum Beispiel die Komponentenmengen für die gerade noch Berechnungen ausgeführt werden (oder eine Kodierung dieser) mit abgespeichert werden. Auch könnte die Ausführung von weiteren Berechnungen für unterschiedliche Komponentenanzahlen durch zusätzliches Abspeichern der noch auf dem Berechnungspfad zu betrachtenden Anzahl von Komponenten *numCompsSplit* entschieden werden. Neben diesen Ansätzen sind noch weitere Ansätze denkbar.

War das in einem Rekursionsschritt des Algorithmus zu berechnende Ergebnis nicht in der Ergebnistabelle gespeichert, wird der Rückgabewert des Rekursionsschritts durch weitere rekursive Aufrufe des Algorithmus berechnet. Dazu wird zuerst die tatsächliche Variable *mappedVariable* zur reduzierten Variablen des Wurzelknotens des BDDs R des ETLEBDDs ermittelt. Falls der Wurzelknoten des BDDs R kein Terminalknoten ist, erfolgt dies durch Auswahl des entsprechenden Eintrags zur durch den Wert von *compIndex* angegebenen Komponente aus der k-fachen Zuordnungsliste des ETLEBDDs (siehe Zeile 11 in Algorithmus 14). Das Ermitteln der tatsächlichen Variablen zu Knoten des BDDs R des ETLEBDDs ist notwendig, um diese Knoten korrekt mit den entsprechenden Knoten des BDDs S über den tatsächlichen Variablen verknüpfen zu können. Ist der BDD R des ETLEBDDs ein Terminalknoten wird *mappedVariable* auf einen Wert, der einen Terminalknoten anzeigt (*TerminalValue*, siehe Zeile 13), gesetzt. Anschließend wird in Zeile 14 des Algorithmus die Top-Variable des aktuellen Rekursionsschritts aus der zum

Wurzelknoten des BDDs R ermittelten tatsächlichen Variablen und der Variablen des Wurzelknotens des BDDs S bestimmt. Die Aufspaltung in Teilprobleme erfolgt im Algorithmus später mit Kofaktoren der BDDs R und S bezüglich Belegungen der ermittelten Top-Variable v mit den beiden Booleschen Werten. Da der BDD R des ETLEBDDs nicht über den tatsächlichen Variablen, sondern über einer reduzierten Variablenmenge definiert ist, müssen Kofaktoren dieses BDDs bezüglich den der tatsächlichen Top-Variable entsprechenden reduzierten Variablen gebildet werden. Neue Kofaktoren des BDDs R müssen aber nur berechnet werden, wenn die zum Wurzelknoten des BDDs R ermittelte tatsächliche Variable der Top-Variable des Rekursionsschritts entspricht (siehe Zeile 15).

Die entsprechenden Kofaktoren bezüglich der reduzierten Variablen w des Wurzelknotens des BDDs R, R_{w_0} bzw. R_{w_1}, werden in den Zeilen 16 bis 18 des Algorithmus berechnet. Andernfalls korrespondiert der Wurzelknoten des BDDs R zu einer später in der Variablenordnung vorkommenden und zu berücksichtigenden tatsächlichen Variablen oder der Wurzelknoten ist ein Terminalknoten. In beiden Fällen wird der BDD R als Übergabeparameter für die anschließend ausgeführten rekursiven Aufrufe verwendet ($R_{w_1} = R_{w_0} = R$, siehe Zeile 20). Nach der Ermittlung der zu verwendenden Kofaktoren des BDDs R, kann die weitere Berechnung des relationalen Produkts durch den Algorithmus auf zwei verschiedene Arten erfolgen. Welche dieser Berechnungen auszuführen ist hängt davon ab, ob im aktuellen Rekursionsschritt für alle Komponenten, für die noch Berechnungen stattfinden, die gleichen Berechnungen durchzuführen sind. Es sind für alle Komponenten die gleichen Berechnungen auszuführen, wenn die Top-Variable des Rekursionsschritts für die Komponenten die gleiche Interpretation besitzt. Dann ist keine Aufspaltung in für Komponenten unterschiedliche Berechnungen nötig. In diesem Fall wird die Funktion *StraightExecution()* (siehe Zeile 22), die in Algorithmus 15 angegeben ist, aufgerufen. In Zeile 21 sind in Algorithmus 14 die Bedingungen angegeben, bei denen keine Aufspaltung der Berechnung in unterschiedliche Berechnungen nötig ist und bei deren Gültigkeit die Funktion *StraightExecution()* aufgerufen wird. Die Funktion wird aufgerufen, wenn der Algorithmus bereits in seinem Ausführungsmodus *iso* ist, da in diesem Ausführungsmodus nur noch für eine Komponente Berechnungen ausgeführt werden. Ebenfalls ist keine Aufspaltung der Berechnung nötig und die Funktion *StraightExecution()* wird aufgerufen, wenn die Top-Variable keine Variable zur Kodierung der Änderungen des lokalen Zustands einer Komponente ist. In diesem Fall ist die Top-Variable eine Variable zur Kodierung von Änderungen des globalen Zustands, für die die Interpretation und die in einem Rekursionsschritt auszuführenden Berechnungen für alle Komponenten gleich sind. Auch sind die gleichen Berechnungen auszuführen, wenn die Top-Variable eine Variable zur Kodierung der Änderungen des lokalen Zustands einer Komponente ist, für die auf dem aktuellen Berechnungspfad aber bereits vorher in eine isolierte Berechnung (im Ausführungsmodus *iso*) verzweigt wurde. Eine Verzweigung in den Ausführungsmodus *iso* des Algorithmus ist für jede Komponente auf einem Berechnungspfad nur einmal, nämlich beim ersten Auftreten einer Variablen zur Kodierung von Änderungen des lokalen Zustands der Komponente als Top-Variable, durchzuführen. Anschließend werden die gesamten weiteren Berechnungen für diese Komponente auf dem Berechnungspfad im Ausführungsmodus *iso* des Algorithmus ausgeführt. Vor dem Wechsel in den Ausführungsmodus *iso* für eine Komponente,

wird der entsprechende Eintrag im Array *previousSplitComp* auf den Wert *TRUE* gesetzt (siehe später eingeführte Funktion *ExecutionSplit()* in Algorithmus 16). Der Wert *TRUE* im Arrayeintrag für eine Komponente zeigt an, dass für die Komponente auf dem aktuellen Berechnungspfad bereits der Wechsel des Algorithmus in den Ausführungsmodus *iso* erfolgt ist. Durch Abfrage dieses Werts für die Komponente, für die eine Top-Variable eine Variable zur Kodierung lokaler Zustände ist, wird auf einem Berechnungspfad der erneute Wechsel in den Ausführungsmodus *iso* für Komponenten, für die dieser Wechsel bereits erfolgt ist, verhindert.

Ist die Top-Variable eines Rekursionsschritts eine Variable zur Kodierung des lokalen Zustands einer Komponente, für die auf dem aktuellen Berechnungspfad noch keine Aufspaltung der Berechnung in den Ausführungsmodus *iso* des Algorithmus erfolgt ist, ist keiner der drei in Zeile 21 angegebenen Fälle erfüllt. Der Algorithmus spaltet die weiteren Berechnungen in einem solchen Rekursionsschritt durch Aufruf der in Algorithmus 16 angegebenen Funktion *ExecutionSplit()* (siehe Zeile 25) auf. Es werden separat weitere Berechnungen für die Komponente, für die die Top-Variable eine Variable zur Kodierung von Änderungen des lokalen Zustands ist, ausgeführt. Außerdem erfolgen Berechnungen für die weiteren noch zu berücksichtigenden Komponenten, die bei mehreren noch vorhandenen restlichen Komponenten für diese weiter gemeinsam durchgeführt werden.

Sind in einem Rekursionsschritt für alle Komponenten die gleichen Berechnungen durchzuführen, ruft der Algorithmus *RelProductETLEBDD()* zur weiteren rekursiven Berechnung des relationalen Produkts die Funktion *StraightExecution()* auf. In dieser Funktion werden rekursive Aufrufe des Algorithmus *RelProductETLEBDD()* für Teilprobleme aus Kofaktoren der BDDs R und S bezüglich Belegungen der Top-Variable mit Booleschen Werten ausgeführt. Ist der Algorithmus *RelProductETLEBDD()* im Ausführungsmodus *iso*, wird als Komponentenindex für die rekursiven Aufrufe in der Funktion *StraightExecution()* der übergebene Komponentenindex *compIndex* verwendet. Dieser gibt die eine Komponente an, für die gerade ausschließlich gültige Berechnungen durchgeführt werden. Wenn die Funktion *StraightExecution()* im Ausführungsmodus *sim* aufgerufen wird, kann es aber notwendig sein, die den rekursiven Aufrufen des Algorithmus *RelProductETLEBDD()* für den Übergabeparameter *compIndex* übergebene Komponente anzupassen. Dies ist in diesem Ausführungsmodus notwendig, wenn die Variable des Wurzelknotens des BDDs R eine Variable zur Kodierung von globalen Zuständen ist und die Variablen der Wurzelknoten des BDDs R_{w_1}, bzw. des BDDs R_{w_0}, Variablen zur Kodierung des lokalen Zustands einer Komponente sind. In diesem Fall kann sich die Komponente, deren Zuordnungsinformationen zur Zuordnung der reduzierten Variablen auf die entsprechenden tatsächlichen Variablen zu verwenden ist, zwischen den BDDs R und R_{w_1} bzw. R_{w_0} unterscheiden. Den Variablen der Wurzelknoten der BDDs R_{w_1} und R_{w_0} ist hier bei den rekursiven Aufrufen die in der Variablenordnung als nächstes vorkommende entsprechende tatsächliche Variable zur Kodierung des lokalen Zustands einer Komponente, für die noch Berechnungen durchgeführt werden müssen, zuzuordnen. Zur Ermittlung der Komponente deren Zuordnungsinformation zu verwenden ist, wird in der Funktion *StraightExecution()* die Funktion *GetLowestComp()* aufgerufen. Diese ist in Algorithmus 19 dargestellt und

wird in Abschnitt 5.4 beschrieben. Die Bestimmung der zu verwendenden Komponente geschieht in der Funktion *GetLowestComp()* unter Benutzung einer Optimierung.

Algorithmus 18: Funktion, die im Algorithmus *RelProductETLEBDD()* aufgerufen wird, um die ermittelten Teilergebnisse R_0 und R_1 zum Ergebnis eines Rekursionsschritts zu kombinieren.

```
1  ResultCalculation(ROBDD R1, ROBDD R0, Variable v, Component compIndex)
2  if R1 == R0 then
3  |  return R1;
4  else
5  |  if IsIdentityPatternVariable(v, compIndex) then
6  |  |  Result = FindOrAddUniqueTable(v, R1, R0);
7  |  else
8  |  |  if IsCurrentStateVar(v) then
9  |  |  |  Result = APPLY(Or, R1, R0);
10 |  |  else
11 |  |  |  Result = FindOrAddUniqueTable(GetCurrentStateVar(v), R1, R0);
12 |  return Result;
```

Nachdem die BDDs der Teilergebnisse R_1 und R_0 durch rekursive Aufrufe des Algorithmus *RelProductETLEBDD()* berechnet wurden, müssen diese zum Resultat des entsprechenden Rekursionsschritts kombiniert werden. Die Funktion *StraightExecution()* ruft dazu die in Algorithmus 18 angegebene Funktion *ResultCalculation()* auf (siehe Zeile 20). Die Funktion *ResultCalculation()* führt die Resultatskombination, wie in Abschnitt 4.3 bei der Vorstellung des Algorithmus zur Berechnung des relationalen Produkts zwischen einem TLEBDD und einem BDD beschrieben, durch. Diese Vorgehensweise ist auch hier korrekt (siehe Abschnitt 5.6). Es wird dort die im relationalen Produkt enthaltene existentielle Quantifizierung bezüglich Variablen für aktuelle Zustände, sowie die Umbenennung von Variablen für Nachfolgerzustände in deren korrespondierende Variablen für aktuelle Zustände, durchgeführt. Die Funktion *IsIdentityPatternVariable()* besitzt zusätzlich zur Top-Variable des aktuellen Rekursionsschritts den aktuell zur Zuordnung von reduzierten Variablen auf tatsächliche Variablen verwendeten Komponentenindex als Übergabeparameter. Mit der Funktion *IsIdentityPatternVariable()* wird geprüft, ob die dieser Funktion übergebene Variable eine Variable zur Kodierung von lokalen Zuständen einer anderen Komponente als der übergebenen Komponente ist. Für die übergebene Komponente ist eine solche Variable dann eine Variable zur Kodierung einer Identitätstransformation. Werden nur noch für eine Komponente Berechnungen ausgeführt, zeigt die Variable *compIndex* auf diese Komponente. Ansonsten wird die Variable *compIndex* immer auf die Komponente mit der höchsten in der Variablenordnung zu einer reduzierten Variablen vorkommende tatsächliche Variable, für die noch keine Aufspaltung in eine separate Berechnung ausgeführt wurde, gesetzt. Die Funktion *StraightExecution()* wird in Rekursionsschritten aufgerufen, in denen keine Aufspaltung der Berechnung in für unterschiedliche Kompo-

nentenmengen gültige Berechnungen erfolgt. Dies ist der Fall, wenn die Top-Variable keine Variable zur Kodierung des lokalen Zustands einer Komponente ist, für die auf dem aktuellen Berechnungspfad noch in eine separate Berechnung verzweigt werden muss. Ist daher in der Funktion *StraightExecution()* die Top-Variable eine Variable zur Kodierung des lokalen Zustands einer Komponente, die ungleich der nächsten für reduzierte Variablen zu benutzenden Komponente *compIndex* ist, dann handelt es sich hier bei der Top-Variablen wenn noch für mehrere Komponenten Berechnungen auszuführen sind für alle Komponenten um eine Variable für eine Identitätstransformation. Zur schnellen Ermittlung ob eine Variable für eine Komponente eine Variable für eine Identitätstransformation ist, können dafür Zusatzinformationen zu den Variablen abgespeichert werden.

Algorithmus 14 führt von den Wurzelknoten des BDDs R des ETLEBDDs und des BDDs S ausgehend Teilberechnungen des relationalen Produkts, die für mehrere Komponenten gleich sind, auf einmal durch. Gleiche Teilberechnungen treten für reduzierte Variablen zur Kodierung des globalen Zustands des Systems auf, die für alle Komponenten auf die gleichen tatsächlichen Variablen abgebildet werden. Außerdem sind zwischen zwei Komponenten die Variablen zur Kodierung von Identitätstransformationen für die Änderungen des lokalen Zustands anderer Komponenten, bis auf unterschiedliche Variablen zur Kodierung der Änderungen des eigenen lokalen Zustands der Komponenten, ebenfalls gleich. Dadurch können für mehrere Komponenten gleiche Teilberechnungen im relationalen Produkt auf einem Berechnungspfad, bis zum Auftreten einer Top-Variable zur Kodierung von lokalen Zuständen einer dieser Komponenten, gemeinsam durchgeführt werden. Eine solche Top-Variable ist nur für eine Komponente eine Variable zur Kodierung des lokalen Zustands, für alle anderen Komponenten stellt die entsprechende tatsächliche Variable eine Variable zur Kodierung einer Identitätstransformation dar. Daher unterscheiden sich in einem solchen Rekursionsschritt die für die eine Komponente und die für die anderen Komponenten für die Top-Variable zu verwendende Interpretation. Aus diesem Grund ist im Rekursionsschritt jeweils unterschiedlich vorzugehen und es können sich verschiedene Ergebnisse bei den auszuführenden Berechnungen ergeben. Deshalb führt der Algorithmus, auf ab den Wurzelknoten der im ersten Aufruf des Algorithmus übergebenen BDDs beginnenden Berechnungspfaden, bei Auftreten einer Variablen zur Kodierung des lokalen Zustands einer Komponente als Top-Variable eine Aufspaltung der durchzuführenden Berechnungen durch, falls dafür auf dem aktuellen Berechnungspfad noch keine Aufspaltung der Berechnung und ein damit verbundener Wechsel in den Ausführungsmodus *iso* des Algorithmus stattgefunden hat. Die Berechnung wird in weiterführende Berechnungen für die Komponente, für die die Top-Variable des aktuellen Rekursionsschritts eine Variable zur Kodierung ihres lokalen Zustands ist und in für die anderen noch zu berücksichtigenden Komponenten gültige Berechnungen aufgespalten. Das Aufspalten der weiteren Berechnungen wird im Algorithmus *RelProductETLEBDD()* durch Aufruf der Funktion *ExecutionSplit()* (siehe Zeile 25), die in Algorithmus 16 dargestellt ist, durchgeführt.

In der Funktion *ExecutionSplit()* wird zuerst die Komponente ermittelt, für die die Top-Variable des Rekursionsschritts eine Variable zur Kodierung des lokalen Zustands ist (siehe Zeile 4 in Algorithmus 16). Dies ist notwendig, da diese Komponente von der durch

compIndex angegebenen Komponente abweichen kann, wenn im BDD des ETLEBDDs kein zur Top-Variablen korrespondierender Knoten vorhanden ist. Anschließend wird der Wert der Variablen *RemovedCounter* angepasst (Zeile 5). Diese Variable wird für die Optimierung der Komponentenauswahl verwendet. Eine Beschreibung der Benutzung der Variablen ist im nachfolgenden Abschnitt angegeben. In Zeile 6 wird das bereits zuvor erwähnte Array *previousSplitComp* aktualisiert. Dieses kann als globale Variable implementiert werden und es besitzt für jede Komponente, für die im ETLEBDD Transitionen gespeichert sind, einen Arrayeintrag. In diesem Array wird für Komponenten vermerkt, ob für diese auf einem Berechnungspfad bereits in den Ausführungsmodus *iso* gewechselt wurde. Dies geschieht durch Setzen des Arrayeintrags der entsprechenden Komponente auf den Wert *TRUE*. Durch Abfrage von Einträgen dieses Arrays kann ermittelt werden, für welche Komponenten auf einem Berechnungspfad bereits in den Ausführungsmodus *iso* des Algorithmus verzweigt wurde. Für diese darf bei den weiteren für diesen Berechnungspfad durchgeführten rekursiven Berechnungen nicht mehr in den Ausführungsmodus *iso* gewechselt werden. In Zeile 6 der Funktion *ExecutionSplit()* wird im Arrayeintrag der Komponente, für die die aktuelle Top-Variable eine Variable zur Kodierung von lokalen Zuständen ist (*compIso*) vermerkt, dass für diese nun auf dem aktuellen Berechnungspfad eine Aufspaltung der Berechnung in den Ausführungsmodus *iso* des Algorithmus durchgeführt wurde. Anschließend werden in der Funktion *ExecutionSplit()* zuerst die für Komponente *compIso* im Ausführungsmodus *iso* durchzuführenden rekursiven Aufrufe des Algorithmus ausgeführt. Bei einer Aufspaltung der Berechnung des relationalen Produkts in Teilprobleme aus Kofaktoren bezüglich einer Top-Variablen zur Kodierung von lokalen Zuständen einer Komponente, ergeben sich wie in Satz 5.7 in Abschnitt 5.6 gezeigt wird, unterschiedliche Teilprobleme für Top-Variablen zur Kodierung von aktuellen Zuständen und für Top-Variablen zur Kodierung von Nachfolgerzuständen. Deshalb wird in der Funktion *ExecutionSplit()* eine Fallunterscheidung durchgeführt (siehe Zeile 7), um die rekursiven Aufrufe des Algorithmus für eine Top-Variable zur Kodierung von aktuellen Zuständen (Zeilen 8 und 10) bzw. für eine Top-Variable zur Kodierung von Nachfolgerzuständen korrekt auszuführen (Zeilen 13 und 15).

Wie in Satz 5.7 gezeigt wird, unterscheiden sich die Teilprobleme im als Übergabeparameter zu verwendenden Kofaktor des BDDs S zur Repräsentation einer Zustandsmenge. Im BDD S sind nur Variablen zur Kodierung von aktuellen Zuständen wesentliche Variablen. Daher muss bei einer Top-Variablen für Nachfolgerzustände, im Gegensatz zu einer Top-Variablen für aktuelle Zustände, kein Kofaktor des BDDs S berechnet werden und der BDD S kann unverändert als Übergabeparameter für die rekursiven Aufrufe verwendet werden. Außerdem unterscheidet sich die Berechnung des Resultats eines Rekursionsschritts aus den ermittelten BDDs mit Teilergebnissen R_1 und R_0. Bezüglich Variablen zur Kodierung von aktuellen Zuständen ist im relationalen Produkt die existentielle Quantifizierung durchzuführen. Bei für Variablen für Nachfolgerzustände zu erzeugenden Knoten müssen stattdessen Knoten für die zu diesen korrespondierenden Variablen für aktuelle Zustände gebaut werden. Dadurch erfolgt direkt die im relationalen Produkt enthaltene Umbenennung von Variablen für Nachfolgerzustände in Variablen für aktuelle Zustände. Die Kombination der rekursiv ermittelten Teilergebnisse wird wieder durch Aufruf der

Funktion *ResultCalculation()* berechnet (siehe Zeile 17 in Algorithmus 16 und Algorithmus 18), die bereits zur Kombination der berechneten Teilresultate zum Rückgabewert der Funktion *StraightExecution()* verwendet und dort beschrieben wurde.

Danach erfolgen die rekursiven Aufrufe des Algorithmus *RelProductETLEBDD()* zur weiteren Berechnung des relationalen Produkts für die Komponenten, für die auf dem aktuellen Berechnungspfad noch keine Aufspaltung der Berechnung in den Ausführungsmodus *iso* durchgeführt wurde. Dazu wird zuerst die Komponente ermittelt, deren Zuordnungsliste als nächstes zur Abbildung der reduzierten Variablen von Knoten des BDDs R des ETLEBDDs auf die zu diesen korrespondierenden tatsächlichen Variablen verwendet wird. Der Index dieser Komponente wird in der Variable *newCompIndex* gespeichert. Ist der Wurzelknoten des BDDs R ein Knoten für eine Variable zur Kodierung des globalen Zustands des Systems, dann wird der reduzierten Variablen dieses Knotens durch die Zuordnungsinformation jeder Komponente die gleiche tatsächliche Variable zugeordnet. In diesem Fall setzt der Algorithmus die Variable *newCompIndex* auf die Komponente mit der nächsten in der Variablenordnung vorkommenden tatsächlichen Variable zur Kodierung der lokalen Zustände der Komponente, für die auf dem aktuellen Berechnungspfad noch keine Aufspaltung der Berechnung in den Ausführungsmodus *iso* ausgeführt wurde. Kommen für keine Komponente mehr lokale Variablen in der Variablenordnung vor, wird die Variable *newCompIndex* auf den Index der Komponente 0 gesetzt. Ermittelt wird die entsprechende Komponente durch Aufruf der Funktion *GetNextLocVarComp()* (siehe Zeile 19). Von dieser kann die Komponente mit Hilfe der Variablenordnung und im voraus einmal berechneter Zusatzinformation, die angibt, für welche Komponenten tatsächliche Variablen Variablen zur Kodierung des lokalen Zustands sind, schnell ermittelt werden. Zusätzlich ist noch das Array *previousSplitComp*, mit der Information für welche Komponenten bereits eine Aufspaltung der Berechnung durchgeführt wurde, zu betrachten. Dadurch behalten die im Algorithmus verwendeten Terminalfälle und die Zugriffe auf die Ergebnistabelle, wie oben beschrieben, ihre Gültigkeit. Ist die Top-Variable eine Variable zur Kodierung von lokalen Zuständen, wird wieder die Funktion *GetLowestComp()*, die in Abschnitt 5.4 eingeführt wird, verwendet. Mit dieser wird die Komponente mit der als erstes in der Variablenordnung vorkommenden tatsächlichen Variablen zur Variablen des Wurzelknotens des BDDs R, für die auf dem aktuellen Berechnungspfad noch keine Aufspaltung der Berechnung durchgeführt wurde, ermittelt und zurückgegeben (siehe Zeile 21). Die Anzahl der Komponenten, für die bei der Berechnung des relationalen Produkts auf einem Berechnungspfad bereits in den Ausführungsmodus *iso* gewechselt wurde, wird in der globalen Variablen *numCompsSplit* gespeichert. Diese Variable wird in der Funktion *ExecutionSplit()* vor Ausführung der weiteren rekursiven Aufrufe für die Komponente, für die in den Ausführungsmodus *iso* gewechselt wurde, inkrementiert (Zeile 22). Vor dem Verlassen der Funktion wird der Wert der Variable wieder dekrementiert (Zeile 35), da der in der Funktion für eine Komponente durchgeführte Wechsel in den Ausführungsmodus *iso* nur für Fortsetzungen des aktuellen Berechnungspfads ab dem aktuellen Rekursionsschritt berücksichtigt werden darf. Mit Hilfe der Variablen *numCompsSplit* kann ermittelt werden, für wieviele Komponenten auf dem aktuellen Berechnungspfad bereits der Wechsel in den Ausführungsmodus *iso* erfolgt ist. Damit kann im Algorithmus entschieden

werden, wenn auf einem Berechnungspfad nur noch für eine Komponente Berechnungen durchzuführen sind[1]. Es können die entsprechenden rekursiven Aufrufe des Algorithmus dadurch für die letzte zu betrachtende Komponente auf einem Berechnungspfad im Ausführungsmodus *iso* durchgeführt werden. Der für die rekursiven Aufrufe zu verwendende Ausführungsmodus wird dabei in der Variablen *newMode* gespeichert (siehe Zeilen 26 bis 28).

Korrespondierende Variablenpaare aus einer Variablen für aktuelle Zustände und einer Variablen für Nachfolgerzustände zur Kodierung des lokalen Zustands einer Komponente, sind für die anderen Komponenten Variablen zur Kodierung von Identitätstransformationen. Im Algorithmus *RelProductETLEBDD()* wird die Aufspaltung in Teilprobleme für solche Variablen von Identitätstransformationen hier gleich bezüglich beider Variablen einer Identitätstransformation durchgeführt. Die Korrektheit dieses Vorgehens wird in Satz 5.7 in Abschnitt 5.6 gezeigt. Im BDD R des ETLEBDDs sind bei Interpretation von diesem für eine Komponente für diese Identitätstransformationen aufgrund der impliziten Berücksichtigung dieser keine Knoten vorhanden. Da der BDD S im relationalen Produkt nur über Variablen für aktuelle Zustände definiert ist, sind in diesem immer nur Knoten für Variablen für aktuelle Zustände enthalten. Um die gemeinsame Berücksichtigung von Variablen solcher Identitätstransformationen zu implementieren, wird in der Funktion *ExecutionSplit()* die korrespondierende Variable für aktuelle Zustände z zu einer Top-Variablen für Nachfolgerzustände v in Zeile 25 ermittelt. Bezüglich dieser werden dann die Kofaktoren $S|_{z=1}$ bzw. $S|_{z=0}$ für die Teilprobleme berechnet. Diese werden bei den rekursiven Aufrufen des Algorithmus *RelProductETLEBDD()* für die Berechnungen des zweiten Ausführungspfads des Rekursionsschritts als Übergabeparameter verwendet (siehe Zeile 29 und 31). Als Kofaktor des BDDs des ETLEBDDs wird der BDD R selbst verwendet. Der Grund dafür ist, dass die Variable v für die Komponenten, für die die hier durchzuführende Berechnung gültig ist, eine Variable einer Identitätstransformation ist. Daher ist für diese im BDD R kein korrespondierender Knoten vorhanden. Anschließend erfolgt die Ermittlung des Resultats für diesen Ausführungspfad wieder mit der Funktion *ResultCalculation()* (siehe Zeile 33). Die Resultate für beide Aufsplittungen der Berechnung $Result_{split1}$ und $Result_{split2}$ werden danach durch Veroderung zum Ergebnis des Rekursionsschritts kombiniert. Dies ist möglich, da beide Resultate gültige Ergebnisse für den aktuellen Rekursionsschritt sind, die durch die Veroderung vereinigt werden. Vor Rückgabe des Ergebnisses des Rekursionsschritts finden noch Anpassungen der Werte von Variablen statt, die nur für vom Rekursionsschritt ausgehende Berechnungen gültig waren. Dabei wird unter anderem der Wert der Variablen *RemovedCounter*, die in Zeile 5 inkrementiert wurde, wieder dekrementiert (Zeile 37).

[1] Hier wird davon ausgegangen, dass bei der rekursiven Berechnung des relationalen Produkts auf jedem Berechnungspfad für jede Komponente mindestens ein Knoten mit einer Variablenmarkierung für die Variablen zur Kodierung des lokalen Zustands der Komponente vorhanden ist. Ist dies für eine Komponente nicht der Fall, sind für Änderungen des lokalen Zustands der Komponente auf dem entsprechenden Berechnungspfad alle Kombinationen von Booleschen Werten durch die Transitionsrelation erlaubt. Dies kommt in Systemen gewöhnlich nicht vor. Andernfalls kann der Algorithmus zur Behandlung dieses Sonderfalls angepasst werden.

Das hier beschriebene vom Algorithmus verwendete Vorgehen zur Berechnung des relationalen Produkts zwischen einem ETLEBDD zur Repräsentation einer Menge von Transitionen und einem BDD zur Repräsentation einer Zustandsmenge kann auch zur Berechnung Boolescher Operationen mit ETLEBDDs benutzt werden. Dazu sind unter anderem die Terminalfälle des Algorithmus und die Kombination der in der Funktion *ResultCalculation()* ermittelten Teilresultate entsprechend anzupassen.

5.4 Auswahl der nächsten Komponente im Algorithmus

Im in Abschnitt 5.3 vorgestellten Algorithmus zur kombinierten Berechnung des relationalen Produkts mit einem ETLEBDD wird mehrmals die Funktion *GetLowestComp()* aufgerufen. Die Funktion wird dort benutzt, um zu einem übergebenen Wurzelknoten eines BDDs R eines ETLEBDDs die Komponente zu ermitteln, deren Zuordnungsinformationen als nächstes zur Abbildung von reduzierten Variablen auf die diesen entsprechenden tatsächlichen Variablen verwendet werden soll. Dazu wird in der Funktion *GetLowestComp()* die Komponente bestimmt, für die auf dem aktuellen Berechnungspfad des Algorithmus *RelProductETLEBDD()* noch keine isolierte Berechnung (im Ausführungsmodus *iso* des Algorithmus) durchgeführt wurde und die unter diesen Komponenten die zum übergebenen Wurzelknoten des BDDs R des ETLEBDDs als nächstes in der Variablenordnung vorkommende korrespondierende tatsächliche Variable besitzt.

Die Zuordnungsliste der ermittelten Komponente wird anschließend im Algorithmus *RelProductETLEBDD()* zur Zuordnung der reduzierten Variablen zu den diesen entsprechenden tatsächlichen Variablen verwendet. Pseudocode der Funktion *GetLowestComp()*, die den BDD R und die Zuordnungsliste *mapping* eines ETLEBDDs als Übergabeparameter besitzt, ist in Algorithmus 19 angegeben. Zur Bestimmung der Komponente mit den oben angegebenen Eigenschaften, wird in der Funktion *GetLowestComp()* eine Optimierung verwendet. Diese zielt darauf ab, das Betrachten der der reduzierten Variablen des Wurzelknotens des BDDs R entsprechenden tatsächlichen Variablen für alle Komponenten zu vermeiden. Aufgrund der bei BDDs verwendeten Reduktionsregeln können auf Pfaden im BDD R des ETLEBDDs oder im BDD S Knoten für Variablen fehlen. Dadurch kann die Reihenfolge der nächsten zur Zuordnung der reduzierten Variablen des BDDs R auf die tatsächlichen Variablen auszuwählenden Komponente von der Reihenfolge des Auftretens der nächsten zur Kodierung des lokalen Zustands von Komponenten vorkommenden tatsächlichen Variablen in der Variablenordnung abweichen. Aus diesem Grund können die Komponenten auf Berechnungspfaden nicht einfach gemäß der Reihenfolge des Vorkommens der tatsächlichen Variablen zur ersten reduzierten Variablen zur Kodierung des lokalen Zustands von Komponenten in der Variablenordnung ausgewählt werden. Die Funktion *GetLowestComp()* nutzt die Reihenfolge des Auftretens von tatsächlichen Variablen zur Kodierung des lokalen Zustands von Komponenten in der Variablenordnung aber dennoch aus. Damit wird versucht das Betrachten der Zuordnungsinformationen aller Komponenten zur Ermittlung der für einen Knoten des BDDs R zu verwendenden Zuordnungsliste zu vermeiden. Dazu wird zu einer gegebenen Variablenordnung die Reihenfolge der Komponenten, in der tatsächliche Variablen zur Kodierung

Algorithmus 19: Funktion zur Bestimmung einer Komponente, für die auf dem aktuellen Berechnungspfad noch keine isolierte Berechnung durchgeführt wurde. Es wird die solche Komponente ermittelt, für die dem übergebenen Wurzelknoten des BDDs R des ETLEBDDs die höchste tatsächliche Variable in der Variablenordnung zugeordnet wird.

```
1  GetLowestComp(ETLEBDD_BDD R, MappingList mapping)
2  positionLocalVarPair = GetPosition(R);
3  if positionLocalVarPair<savedVarCompOrders then
4  |  removeCount = removedCounter[positionLocalVarPair];
5  |  for i = removeCount;i < numberOfComponents;i++ do
6  |  |  if previousSplitComp[varCompOrder[positionLocalVarPair][i]]==FALSE
   |  |  then
7  |  |  |  newCompIndex = varCompOrder[positionLocalVarPair][i];
8  |  |  |  break;
9  else
10 |  lowestVarLevel = numberOfVariables;
11 |  newCompIndex = 0;
12 |  for i = 0;i < numberOfComponents;i++ do
13 |  |  if previousSplitComp[i]==FALSE &&
14 |  |  |  mapping[i][VariableRoot(R)]<lowestVarLevel then
15 |  |  |  newCompIndex = i;
16 |  |  |  lowestVarLevel = mapping[i][VariableRoot(R)];
17 return newCompIndex;
```

des lokalen Zustands von Komponenten in der Variablenordnung vorkommen, abgespeichert. Da korrespondierende Variablen für aktuelle Zustände und Nachfolgerzustände in verschachtelten Variablenordnungen immer benachbart sind, ist diese Information für beide Variablen eines solchen Variablenpaares gleich. Das Abspeichern muss daher für ein korrespondierendes Variablenpaar nur einmal durchgeführt werden. Für die Funktion *GetLowestComp()* wird die Anzahl der Variablenpaare, für die die Reihenfolge des Vorkommens von Variablen für lokale Zustände in der Variablenordnung vorher abgespeichert wird, durch die globale Variable *savedVarCompOrders* angegeben. Bei den in dieser Arbeit durchgeführten Verifikationsexperimenten wurde die Variable *savedVarCompOrders* auf den Wert 3 gesetzt. Die Reihenfolge des Vorkommens von tatsächlichen Variablen zur Kodierung des lokalen Zustands von Komponenten in der Variablenordnung wird im Array *varCompOrder* abgespeichert. Da für ein korrespondierendes Variablenpaar zur Speicherung dieser Information nur ein Arrayeintrag notwendig ist, wird die Reihenfolge im Array *varCompOrder* für drei Variablenpaare abgespeichert. Das Array *varCompOrder* ist zweidimensional. In der ersten Dimension ist ein Arrayeintrag für jedes Variablenpaar aus reduzierten Variablen, für das Informationen gespeichert werden, vorhanden. Die zweite Dimension speichert die Reihenfolge des Vorkommens der zu den reduzierten Variablen korrespondierenden tatsächlichen Variablen der Komponenten in der verwendeten Variablenordnung. Dazu wird im i-ten Arrayeintrag der zweiten Dimension die Komponente, für die das Variablenpaar als i-tes in der Variablenordnung vorkommt, abgespeichert. In der vorliegenden Arbeit wurden in Verifikationsexperimenten nur Systeme mit replizierten Komponenten verwendet und es wurden reduzierte Variablen für lokale Zustände für alle Komponenten immer den sich entsprechenden tatsächlichen Variablen zur Kodierung des lokalen Zustands der Komponenten zugeordnet. Bei Ausführung der Funktion *GetLowestComp()* wird zuerst ermittelt, zur Kodierung des wievielten Variablenpaares zur Kodierung von lokalen Zuständen in der Variablenordnung der reduzierten Variablen, die reduzierte Variable des Wurzelknotens des BDDs R benutzt wird. Dies geschieht durch Aufruf der Funktion *GetPosition()* (siehe Zeile 2 in Algorithmus 19) und kann nach vorherigem Abspeichern dieser Informationen direkt bestimmt werden. Ist die Reihenfolge des Vorkommens der zu den reduzierten Variablen korrespondierenden tatsächlichen Variablen der Komponenten in der Variablenordnung für das ermittelte Variablenpaar noch abgespeichert (d.h. es gilt bei 3 Arrayeinträgen für Variablenpaare im Array *varCompOrder* (savedVarCompOrders=3) und Verwendung des Werts 0 für das erste Variablenpaar positionLocalVarPair<savedVarCompOrders, siehe Zeile 3), kann das Betrachten der Zuordnungsinformationen aller Komponenten vermieden werden. Dabei wird in der Funktion *GetLowestComp()* eine weitere Optimierung verwendet. Wird für eine Komponente in der Funktion *ExecutionSplit()* in den Ausführungsmodus *iso* des Algorithmus verzweigt, so müssen in der Sortierung für das entsprechende Variablenpaar zur Top-Variablen des Rekursionsschritts nur noch nach dieser Komponente vorkommende Komponenten betrachtet werden. Daher kann der nächste Zugriff auf dieses Array mit dem nachfolgenden Arrayeintrag gestartet werden. Um dies auszunutzen, wird ein Array *removedCounter* verwendet. Dieses besitzt einen Arrayeintrag für jedes Variablenpaar, für das die Reihenfolge der Komponenten gespeichert wurde. Der Arrayeintrag des Variablenpaares zur Top-Variablen v

wird vor dem Wechsel in den Ausführungsmodus *iso* des Algorithmus in der Funktion *ExecutionSplit()* inkrementiert (siehe Zeile 5 in Algorithmus 16). Zur Ermittlung des als nächstes zu verwendenden Komponentenindexes, können die Einträge des Arrays *varCompOrder* für ein Variablenpaar dadurch beginnend mit dem durch die Variable *removeCount* angegebene Eintrag betrachtet werden. Bei Auffinden der ersten Komponente, für die auf dem aktuellen Berechnungspfad noch keine Aufspaltung der Berechnung des Algorithmus *RelProdETLEBDD()* in den Ausführungsmodus *iso* durchgeführt wurde, wurde die passende Komponente in der Sortierung gefunden (siehe Zeilen 7 und 8). Zur korrekten Benutzung des Arrays *removedCounter*, müssen in einem Rekursionsschritt inkrementierte Arrayeinträge nach Berechnung des Resultats des Rekursionsschrittes wieder dekrementiert werden. Dies ist notwendig, da eine Inkrementierung eines Eintrags dieses Arrays nur für Fortsetzungen des aktuellen Berechnungspfads gültig ist. Dazu wurde ein Aufruf der Funktion *DecrementRemovedCounter()* in die Funktion *ExecutionSplit()* eingefügt (siehe Zeile 37 in Algorithmus 16). In der Funktion wird geprüft, ob für das Variablenpaar mit der Top-Variablen v die Reihenfolge des Vorkommens des Variablenpaares für Komponenten im Array *varCompOrder* gespeichert ist. Falls ja, wird der entsprechende Arrayeintrag des Arrays *removedCounter* dekrementiert. Bei den in dieser Arbeit durchgeführten Verifikationsexperimenten konnte die zur Zuordnung auf tatsächliche Variablen zu verwendende Komponente bei Aufruf der Funktion *GetLowestComp()* nahezu ausnahmslos mit der Vorsortierung für das erste Variablenpaar gefunden werden. War dies nicht möglich, wurde die Komponente immer mit Hilfe der drei für Variablenpaare gespeicherten Reihenfolgen von Komponenten ermittelt. Ist für ein Variablenpaar keine Komponentenreihenfolge vorabgespeichert, wird in der Funktion *GetLowestComp()* die Zuordnung der reduzierten Variablen des Wurzelknotens des BDDs R zur entsprechenden tatsächlichen Variablen für alle Komponenten betrachtet. Anschließend wird die Komponente mit der als erstes in der Variablenordnung vorkommenden tatsächlichen Variablen ausgewählt, für die auf dem aktuellen Berechnungspfad noch keine Aufspaltung der Berechnung in den Ausführungsmodus *iso* des Algorithmus durchgeführt wurde. Dies ist in Algorithmus 19 in den Zeilen 10 bis 16 dargestellt.

5.5 Beispielablauf des Algorithmus

In diesem Abschnitt wird das Vorgehen des in Abschnitt 5.3 vorgestellten Algorithmus bei der simultanen kombinierten Berechnung des relationalen Produkts mit einem ETLEBDD für mehrere Komponenten veranschaulicht. Dazu sind in Abbildung 5.1 zum in Abschnitt 2.6 präsentierten Beispielsystem ein ETLEBDD zur Repräsentation von Transitionen der zwei Komponenten des Beispielsystems (verwendete Variablenordnung $<_{level_4}$, mit $y_g <_{level_4} y'_g <_{level_4} y_1 <_{level_4} y'_1$) und ein BDD zur Repräsentation des Anfangszustands des Systems (verwendete Variablenordnung $<_{level_3}$, mit $x_g <_{level_3} x'_g <_{level_3} x_1 <_{level_3} x'_1 <_{level_3} x_2 <_{level_3} x'_2$) angegeben. Der ETLEBDD repräsentiert die Transitionen der Menge $T = \{((0, s_0, s_0),(0, s_1, s_0)), ((0, s_0, s_1),(0, s_1, s_1)), ((0, s_0, s_0),(0, s_0, s_1)), ((0, s_1, s_0),(0, s_1, s_1))\}$. Dies ist daran zu sehen, dass beim auf den Terminalknoten für den Booleschen Wert 1 auswertenden Pfad des ETLEBDDs die globale Variable

im aktuellen Zustand (Variable y_g) und im Nachfolgerzustand (Variable y'_g) den Wert 0 besitzt. Außerdem wird mit diesem Pfad die Änderung des lokalen Zustands vom Booleschen Wert 0 (mit dem der Zustand s_0 kodiert wird (siehe Abschnitt 2.6), Variable y_1) zum Booleschen Wert 1 (mit dem der Zustand s_1 kodiert wird, Variable y'_1) repräsentiert. Die beschriebenen Zustandsübergänge sind dabei für beide Komponenten im ETLEBDD gespeichert. Bei Interpretation des BDDs des ETLEBDDs für eine Komponente werden für den Zustandsübergang des lokalen Zustands der anderen Komponente implizit Identitätstransformationen angenommen. Als BDD für den Anfangszustand wird der in Abbildung 2.14 in Abschnitt 2.6 aufgeführte BDD verwendet. Der Ablauf des Algorithmus mit dem ETLEBDD mit Transitionen der beiden Komponenten und dem BDD für den Anfangszustand als Eingabe, wird aus Gründen der besseren Veranschaulichung im Folgenden ohne die Benutzung der Ergebnistabelle beschrieben. Gestartet wird der Algorithmus im Ausführungsmodus *sim*. Der Übergabeparameter *compIndex* zeigt auf Komponente 1 und die Variable *numCompsSplit* besitzt zu Beginn den Wert 0. Die 2-fache Zuordnungsliste *mapping* besteht aus den folgenden Zuordnungslisten der Komponenten:

- Zuordnungsliste Komponente 1: $[x_g,x'_g,x_1,x'_1]$
- Zuordnungsliste Komponente 2: $[x_g,x'_g,x_2,x'_2]$

Im *1. Aufruf* des Algorithmus liegt keiner der in Algorithmus 17 angegebenen Terminalfälle vor und die Variable y_g, die hier die Variable des Wurzelknotens des ETLEBDDs ist, ist eine Variable zur Kodierung des globalen Zustands des Systems. Daher wird sie von der Zuordnungsinformation beider im System vorhandenen Komponenten auf die gleiche tatsächliche Variable x_g abgebildet. Die Variable x_g ist damit für beide Komponenten die tatsächliche Variable zum Wurzelknoten des BDDs des ETLEBDDs. Da sie im BDD mit dem Anfangszustand auch die tatsächliche Variable des Wurzelknotens ist, ist sie die Top-Variable des *1. Aufrufs* des Algorithmus. Weil die Top-Variable x_g eine Variable zur Kodierung von globalen Zuständen ist, besitzt sie für beide Komponenten die gleiche Interpretation. Deshalb ist die im aktuellen Rekursionsschritt bezüglich dieser Top-Variable durchzuführende Aufspaltung in Teilprobleme für beide Komponenten gleich und es ist die in Zeile 21 im Algorithmus *RelProductETLEBDD()* angegebene Bedingung erfüllt (*!IsLocalCompVar(x_g)==TRUE*, da die Variable x_g keine Variable zur Kodierung von lokalen Zuständen einer Komponente ist). Daher werden die weiteren Berechnungen des Algorithmus in diesem Rekursionsschritt ohne Aufspaltung in für die Komponenten unterschiedliche Berechnungen, durch Aufruf der Funktion *StraightExecution()* fortgesetzt.

In der in Algorithmus 15 angegebenen Funktion *StraightExecution()* werden anschließend weitere rekursive Aufrufe des Algorithmus *RelProductETLEBDD()* durchgeführt. Bei diesen werden Kofaktoren bezüglich der Top-Variable x_g, bzw. für den BDD des ETLEBDDs bezüglich der dieser entsprechenden reduzierten Variablen y_g, als Übergabeparameter verwendet ($R|_{y_g=1}$ (R_{w_1}) und $R|_{y_g=0}$ (R_{w_0}) bzw. $S|_{x_g=1}$ und $S|_{x_g=0}$). In der Funktion *StraightExecution()* sind die in den Zeilen 4 und 12 abgefragten Bedingungen nicht erfüllt. Die Wurzelknoten der Kofaktoren des BDDs R (R_{w_1} bzw. R_{w_0}) sind keine Knoten für Variablen zur Kodierung des lokalen Zustands einer Komponente (*IsLocalVarVertex()* gibt jeweils den Wert *FALSE* zurück), sondern ebenfalls ein Knoten zum

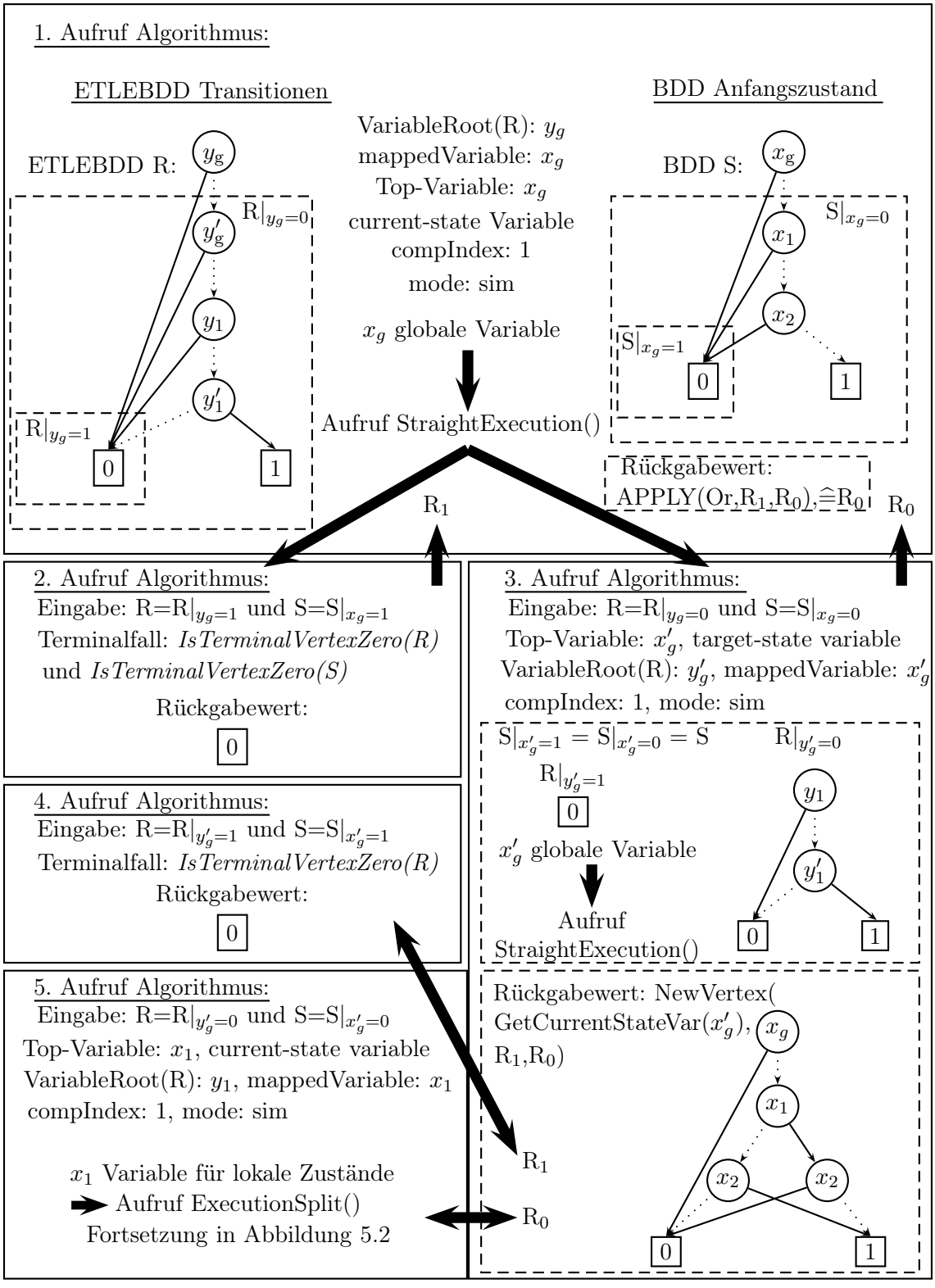

Abbildung 5.1: Beispiel zum Algorithmus zur gemeinsamen Berechnung des relationalen Produkts für mehrere Komponenten mit einem ETLEBDD.

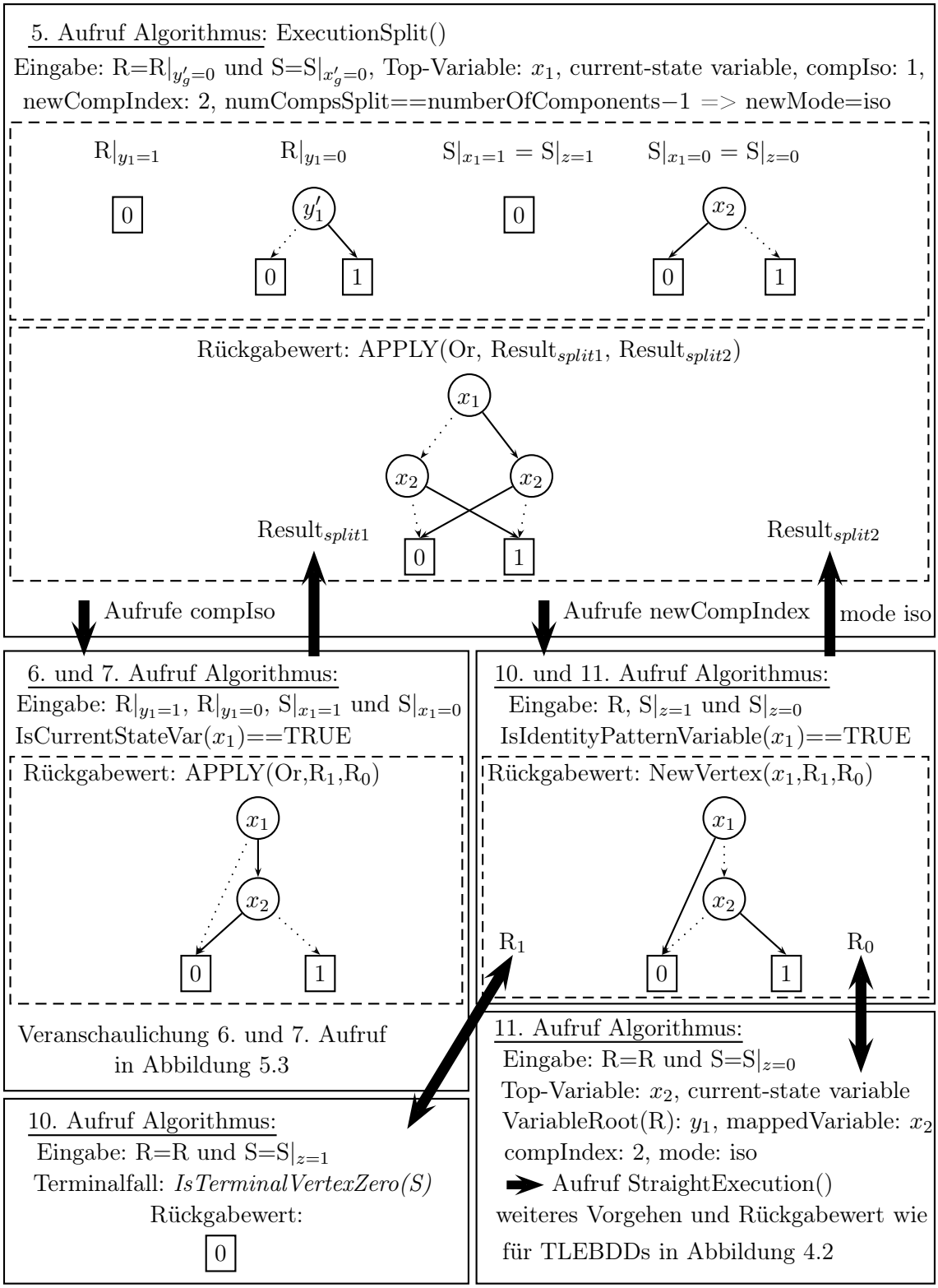

Abbildung 5.2: Veranschaulichung weiterer rekursiver Aufrufe des Algorithmus zur Berechnung des relationalen Produkts mit einem ETLEBDD.

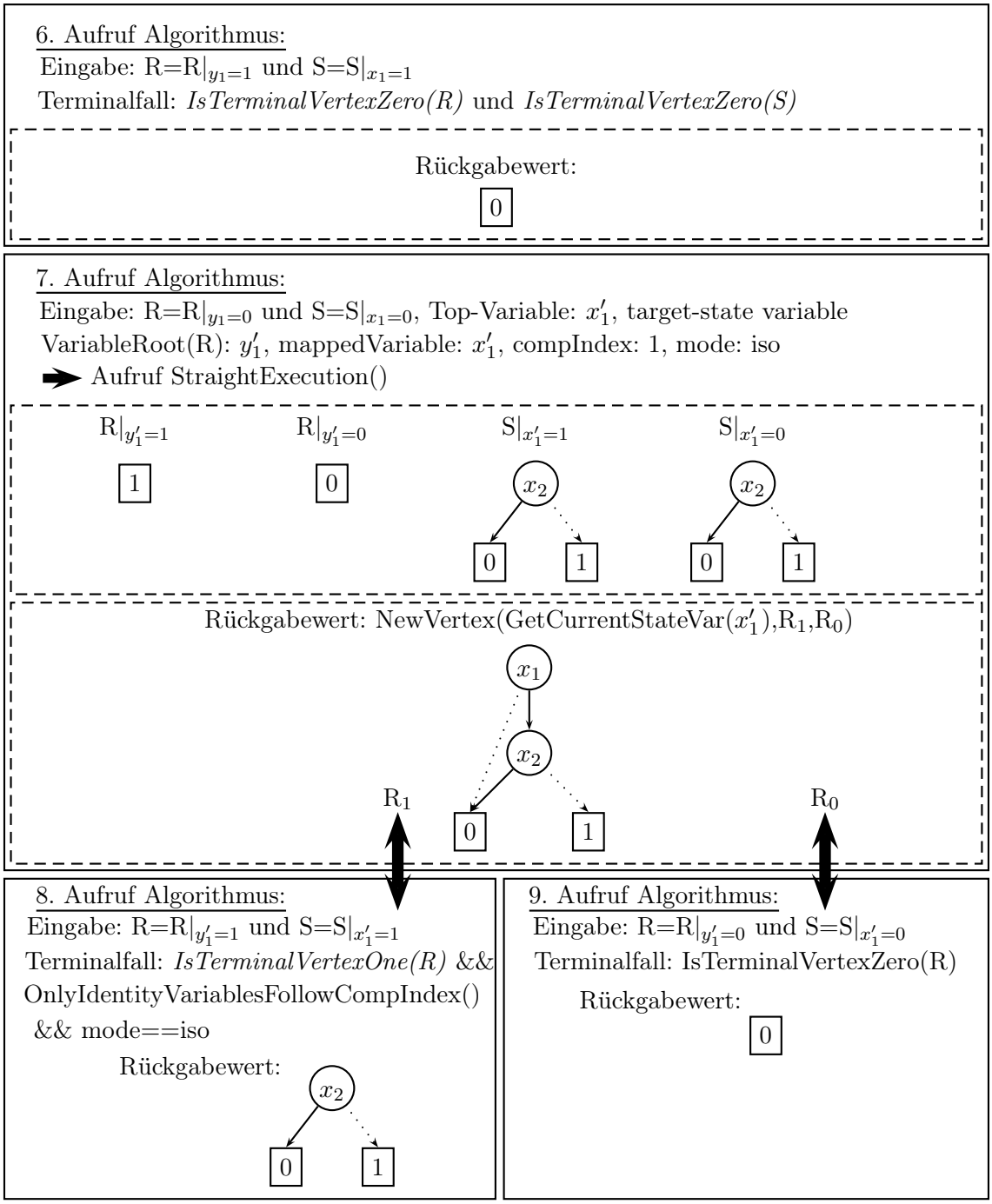

Abbildung 5.3: Veranschaulichung weiterer rekursiver Aufrufe des Algorithmus zur Berechnung des relationalen Produkts mit einem ETLEBDD.

Kodieren von globalen Zuständen (R_{w_0}), bzw. ein Terminalknoten (R_{w_1}). Daher werden in der Funktion *StraightExecution()* die in den Zeilen 10 und 18 der Funktion angegebenen rekursiven Aufrufe des Algorithmus mit den zuvor berechneten Kofaktoren als Übergabeparametern ausgeführt.

Im *2. Aufruf* des Algorithmus, bei dem die Kofaktoren bezüglich des Booleschen Werts 1 der Top-Variable x_g als Übergabeparameter benutzt werden, liegt direkt ein Terminalfall vor. Beide übergebenen Kofaktoren ($R|_{y_g=1}$ und $S|_{x_g=1}$) sind Terminalknoten für den Booleschen Wert 0, weshalb der Rückgabewert des Terminalfalls und dieses Aufrufs des Algorithmus der Terminalknoten für den Booleschen Wert 0 ist (siehe Funktion *TerminalCase()* in Algorithmus 17). Die Top-Variable des *3. Aufrufs* des Algorithmus ist die tatsächliche Variable x'_g, die wieder eine Variable zur Kodierung des globalen Zustands des Systems und die tatsächliche Variable zur reduzierten Variablen y'_g des Wurzelknotens des Kofaktors $R|_{y_g=0}$ ist. Für beide Komponenten wird die reduzierte Variable y'_g durch deren Zuordnungsinformationen auf die tatsächliche Top-Variable x'_g abgebildet. Es wird zur weiteren Berechnung in der Funktion *RelProductETLEBDD()* erneut die Funktion *StraightExecution()* aufgerufen (*!IsLocalCompVar(x'_g)==TRUE*), wodurch der Ausführungsmodus des Algorithmus in beiden in der Funktion *StraightExecution()* durchgeführten rekursiven Aufrufen des Algorithmus *RelProductETLEBDD()* der Modus *sim* bleibt. Mit den Kofaktoren bezüglich des Booleschen Werts 1 der Top-Variablen x'_g als Übergabeparametern ($R|_{y'_g=1}$ (R_{w_1}) und $S|_{x'_g=1}$) wird der *4. Aufruf* des Algorithmus in Zeile 10 der Funktion *StraightExecution()* ausgeführt, da die in Zeile 4 der Funktion angegebene Bedingung nicht erfüllt ist (*IsLocalVarVertex(*$R|_{y'_g=1}$*)==FALSE*). Der Wurzelknoten des BDDs $R|_{y'_g=1}$ zur Repräsentation des entsprechenden Kofaktors ist ein Terminalknoten für den Wert 0. Daher liegt im *4. Aufruf* des Algorithmus direkt ein Terminalfall mit dem Booleschen Wert 0 als Rückgabewert vor. Da der Wurzelknoten des BDDs für den Kofaktor $R|_{y'_g=0}$ (R_{w_0}) ein Knoten zur Kodierung eines lokalen Zustands ist (*IsLocalVarVertex(*$R_{y'_g=0}$*)==TRUE*), ist für die Kofaktoren bezüglich des Booleschen Werts 0 der Top-Variable x'_g die in Zeile 12 in der Funktion *StraightExecution()* angegebene Abfrage erfüllt. Daher wird zur weiteren rekursiven Berechnung der in Zeile 15 in dieser Funktion angegebene rekursive Aufruf verwendet. Zuvor wird die Variable *newCompIndex* auf den Index der Komponente mit der höchsten in der Variablenordnung vorkommenden tatsächlichen Variablen zum Wurzelknoten des BDDs des ETLEBDDs, für die auf dem aktuellen Berechnungspfad noch keine Aufspaltung in den Ausführungsmodus *iso* ausgeführt wurde, gesetzt (Zeile 14). Die entsprechende Komponente ist hier Komponente 1.

Der korrespondierende *5. Aufruf* des Algorithmus ist der erste Aufruf des Algorithmus *RelProductETLEBDD()* mit einer Variablen zur Kodierung des lokalen Zustands einer Komponente als Top-Variablen. In diesem Rekursionsschritt ist der Wurzelknoten des BDDs des ETLEBDDs ein Knoten für die reduzierte Variable y_1. Für Komponente 1 ist diese Variable eine Variable zur Kodierung von lokalen Zuständen der Komponente. Dieser reduzierten Variablen wird von der Zuordnungsinformation der Komponente die tatsächliche Variable x_1 zugeordnet. Bei Komponente 2 wird die reduzierte Variable y_1 stattdessen auf die tatsächliche Variable x_2 abgebildet, da diese für Komponente 2 die der reduzierten Variablen entsprechende tatsächliche Variable zur Kodierung von lokalen Zuständen der

Komponente ist. Der Übergabeparameter *compIndex* besitzt im *5. Aufruf* des Algorithmus *RelProductETLEBDD()* den im aufrufenden Rekursionsschritt ermittelten Wert 1 für Komponente 1. Damit wird der reduzierten Variablen y_1, die dieser für Komponente 1 entsprechende tatsächliche Variable x_1 in Zeile 11 des Algorithmus *RelProductETLEBDD()* korrekt zugeordnet und der Variablen *mappedVariable* zugewiesen.

Die Variable x_1 ist ebenfalls die Top-Variable des Rekursionsschritts. Da diese für Komponente 1 eine Variable zur Kodierung von lokalen Zuständen der Komponente und für Komponente 2 eine Variable zur Kodierung einer Identitätstransformation ist, unterscheiden sich die für die Komponenten in diesem Rekursionsschritt durchzuführenden Berechnungen. Die in Algorithmus *RelProductETLEBDD()* in Zeile 21 angegebene *if*-Abfrage ist nicht erfüllt, da es sich bei der Top-Variablen x_1 für Komponente 1 um eine Variable zur Kodierung des lokalen Zustands handelt (*IsLocalCompVar*(x_1)$==$*TRUE*) und für diese Komponente auf dem aktuellen Berechnungspfad noch keine Aufspaltung der Berechnung in den Ausführungsmodus *iso* des Algorithmus erfolgt ist (*previousSplitComp*[*Komponente 1*]$==$*FALSE*). Daher wird zur weiteren Berechnung in Zeile 25 die in Algorithmus 16 angegebene Funktion *ExecutionSplit()* aufgerufen. In dieser wird zuerst die Berechnung des relationalen Produkts für Komponente 1 durch rekursive Aufrufe im Ausführungsmodus *iso* des Algorithmus fortgesetzt. Da es sich bei der Top-Variablen x_1 um eine Variable für aktuelle Zustände handelt, werden dort die in der Funktion *ExecutionSplit()* in Zeile 8 (*6. Aufruf*) und Zeile 10 (*7. Aufruf*) angegebenen rekursiven Aufrufe des Algorithmus ausgeführt. Die in diesen rekursiven Aufrufen durchgeführten weiteren Berechnungen sind nur noch für Komponente 1 gültig und werden in den Abbildungen 5.2 und 5.3) veranschaulicht. Im *6. Aufruf* des Algorithmus tritt dabei direkt ein Terminalfall mit dem Terminalknoten für den Booleschen Wert 0 als Rückgabewert auf, da beide beim Aufruf als Übergabeparameter übergebenen BDDs ($R|_{y_1=1}$ und $S|_{x_1=1}$) Terminalknoten für den Booleschen Wert 0 sind. Beim *7. Aufruf* des Algorithmus, der die BDDs $R|_{y_1=0}$ und $S|_{x_1=0}$ als Übergabeparameter besitzt und der im Ausführungsmodus *iso* des Algorithmus durchgeführt wird, wird die Zuordnungsliste von Komponente 1 zur Zuordnung auf die tatsächlichen Variablen verwendet. Die Top-Variable des entsprechenden Rekursionsschritts ist die Variable x'_1, die die tatsächliche Variable zur Variablen des Wurzelknotens des BDDs $R|_{y_1=0}$ ist. Da sich der Algorithmus in seinem Ausführungsmodus *iso* befindet, erfolgt die weitere rekursive Berechnung in diesem Rekursionsschritt durch Aufruf der Funktion *StraightExecution()* in Zeile 22 der Funktion *RelProductETLEBDD()*. Ebenfalls aufgrund des Ausführungsmoduses *iso* werden in der Funktion *StraightExecution()* die in den Zeilen 10 und 18 angegebenen rekursiven Aufrufe durchgeführt. In diesen beiden rekursiven Aufrufen, dem *8. Aufruf* und dem *9. Aufruf* des Algorithmus, liegen direkt Terminalfälle vor. Die aufgetretenen Terminalfälle und deren Rückgabewerte sind in Abbildung 5.3 unten angegeben und werden im *7. Aufruf* des Algorithmus zum Rückgabewert dieses Aufrufs kombiniert. Die Top-Variable x'_1 des *7. Aufrufs* des Algorithmus ist eine Variable für Nachfolgerzustände. Für diese wird in diesem Rekursionsschritt bei der Resultatskombination direkt die notwendige unmittelbare Umbenennung in die korrespondierende Variable für aktuelle Zustände durchgeführt. Dazu wird die Funktion *ResultCalculation()* (siehe Algorithmus 18) aufgerufen. In dieser wird dann ein neuer Knoten für die Variable

für aktuelle Zustände x_1, mit den Rückgabewerten des *8.* und *9. Aufrufs* des Algorithmus als Söhnen, erzeugt (*NewVertex(GetCurrentStateVar(x'_1), R_1, R_0)*).

Die Top-Variable x_1 des *5. Aufrufs* des Algorithmus ist eine Variable für aktuelle Zustände. In der in diesem Rekursionsschritt aufgerufenen Funktion *ExecutionSplit()* werden die im *6.* und *7. Aufruf* des Algorithmus ermittelten Rückgabewerte in Zeile 17 anschließend zum Ergebnis der Berechnungen des Algorithmus für Komponente 1 im Ausführungsmodus *iso* kombiniert. Dies erfolgt wieder durch Aufruf der Funktion *ResultCalculation()*. Da es sich bei der Top-Variablen des Rekursionsschritts x_1 um eine Variable für aktuelle Zustände handelt, werden dazu die ermittelten Teilresultate verodert (*APPLY(Or, R_1, R_0)*). Nach Ermittlung des Ergebnisses der rekursiven Berechnung des Algorithmus für Komponente 1 im Ausführungsmodus *iso*, werden im aktuellen Rekursionsschritt weitere Berechnungen für Komponente 2 ausgeführt. Komponente 2 ist die einzige im Beispielsystem vorhandene weitere Komponente. Daher gilt in Zeile 27 der Funktion *ExecutionSplit() numCompsSplit==numberOfComponents*-1 und die weiteren rekursiven Aufrufe der Funktion *RelProductETLEBDD()* für Komponente 2 werden ebenfalls im Ausführungsmodus *iso* des Algorithmus ausgeführt. Im dazu in Zeile 29 in der Funktion *ExecutionSplit()* getätigten *10. Aufruf* des Algorithmus liegt direkt ein Terminalfall der rekursiven Berechnung vor. Der Kofaktor $S|_{x_1=1}$ ($S|_{z=1}$) entspricht dem Terminalknoten für den Booleschen Wert 0, was zum Booleschen Wert 0 als Rückgabewert des Terminalfalls führt. Anschließend wird in Zeile 31 der Funktion *ExecutionSplit()* der *11. Aufruf* des Algorithmus ausgeführt. Die dort durchzuführenden Berechnungen gleichen denen im Verifikationsbeispiel zum Algorithmus zur kombinierten Berechnung des relationalen Produkts mit einem TLEBDD (siehe Abschnitt 4.3 und Abbildung 4.2). Nach dem dort beschriebenen Vorgehen wird die weitere Berechnung des relationalen Produkts auf diesem Berechnungspfad durch den Algorithmus *RelProductETLEBDD()* in dessen Ausführungsmodus *iso* durchgeführt. Der Rückgabewert des *10.* und *11. Aufrufs* des Algorithmus *RelProductETLEBDD()* wird anschließend in Zeile 33 der im *5. Aufruf* des Algorithmus aufgerufenen Funktion *ExecutionSplit()*, durch Aufruf der Funktion *ResultCalculation()* zum für Komponente 2 ermittelten Resultat kombiniert. Da es sich bei der Top-Variablen x_1 dieses Rekursionsschritts für Komponente 2 um eine Variable zur Kodierung einer Identitätstransformation handelt, wird dazu ein neuer Knoten mit den Teilresultaten R_1 und R_0 als Söhnen erzeugt (*NewVertex(x_1, R_1, R_0)*). Die im *5. Aufruf* für beide Komponenten durch die separate Berechnung erhaltenen Rückgabewerte werden anschließend durch Veroderung zum Rückgabewert des *5. Aufrufs* des Algorithmus kombiniert (*APPLY(Or, $Result_{split1}$, $Result_{split2}$)*, siehe Zeile 34 in der Funktion *ExecutionSplit()*). Dies ist korrekt, da beide Teilergebnisse gültige Ergebnisse für die ab dem *5. Aufruf* des Algorithmus durchzuführenden weiteren Berechnungen sind. Anschließend werden die Rückgabewerte des *4.* und *5. Aufrufs* des Algorithmus durch die Resultatskombination (Aufruf Funktion *ResultCalculation()*) für eine Top-Variable für Nachfolgerzustände zum Ergebnis des *3. Aufrufs* des Algorithmus kombiniert. Das resultierende Ergebnis entspricht auch dem Rückgabewert des Algorithmus zur simultanen Berechnung des relationalen Produkts für beide Komponenten und das angegebene Beispiel. Im *1. Aufruf* des Algorithmus ist zur Resultatskombination die Veroderung (*APPLY(Or, R_1, R_0)*) der Rückgabewerte des *2.*

und *3. Aufrufs* des Algorithmus durchzuführen. Da der Rückgabewert des *2. Aufrufs* ein Terminalknoten für den Booleschen Wert 0 ist, ergibt sich dadurch keine Veränderung des Rückgabewerts.

5.6 Korrektheit des Algorithmus

In diesem Abschnitt wird die Korrektheit des Vorgehens des in Abschnitt 5.3 vorgestellten Algorithmus bei der rekursiven Aufspaltung der Berechnung des relationalen Produkts mit einem ETLEBDD in Teilprobleme gezeigt. Im Algorithmus werden verschiedene Aufspaltungen der in einem Rekursionsschritt durchzuführenden Berechnung in Teilprobleme verwendet. Die Korrektheit dieser wird nachfolgend in Satz 5.7 gezeigt. Im Beweis zu diesem Satz wird zur Zuordnung der Variablenindizes von tatsächlichen Variablen auf die zu diesen korrespondierenden Variablenindizes für reduzierte Variablen, wie bei TLEBDDs, die Umkehrfunktion $\pi_i^{-1}(\upsilon)$ der in Definition 4.1 in Abschnitt 4.1 eingeführten Zuordnungsfunktion π_i verwendet. Aus Gründen der besseren Verständlichkeit bekommt die Umkehrfunktion hier als Eingabeparameter ebenfalls anstatt des Variablenindexes einer tatsächlichen Variable, eine tatsächliche Variable übergeben. Sie ermittelt dazu den Variablenindex der korrespondierenden reduzierten Variablen.

Satz 5.7. *Gegeben sei das relationale Produkt mit einem ETLEBDD zur Repräsentation einer Menge isomorpher Transitionen von k Komponenten ($k \geq 2$)* $\exists\vec{x}[(\Delta(IdVar_1, R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_1}\}) \vee ... \vee \Delta(IdVar_k, R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_k}\})) \wedge S(\vec{x})]\{\vec{x}' \leftarrow \vec{x}\}$*, wie es in Abschnitt 5.3 eingeführt wurde. Sei außerdem die aus den Indizes der k-Komponenten bestehende Menge $M = \{1, 2, ..., k\}$ gegeben. Dann sind für das relationale Produkt mit einem ETLEBDD folgende Aufspaltungen bezüglich der Belegungen Boolescher Variablen korrekt:*

- *Für eine tatsächliche Variable υ zur Kodierung des globalen Zustands eines Systems ($\in X$ oder X'), lässt sich das relationale Produkt mit einem ETLEBDD aufspalten in (für die Zuordnungsinformationen zu einer reduzierten Variablen y_p zur Kodierung des globalen Zustands gilt dabei $\forall i, j \in M : \pi_i(y_p) = \pi_j(y_p)$, weshalb in den folgenden beiden Fällen zur besseren Verständlichkeit für solche Variablen eine globale Zuordnung π_g verwendet wird):*
  - *υ tatsächliche Variable für aktuelle Zustände ($\upsilon \in X$):*

$$(\bigvee_{i=1}^{k}(\exists\vec{x}(\Delta(IdVar_i, R(\vec{y})|_{y_{\pi_g^{-1}(\upsilon)}=1}\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}) \wedge S(\vec{x})|_{\upsilon=1}))\{\vec{x}' \leftarrow \vec{x}\}) \vee$$
$$(\bigvee_{i=1}^{k}(\exists\vec{x}(\Delta(IdVar_i, R(\vec{y})|_{y_{\pi_g^{-1}(\upsilon)}=0}\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}) \wedge S(\vec{x})|_{\upsilon=0}))\{\vec{x}' \leftarrow \vec{x}\})$$

  - *υ tatsächliche Variable für Nachfolgerzustände ($\upsilon \in X'$):*

$$((\upsilon)\{\vec{x}' \leftarrow \vec{x}\} \cdot (\bigvee_{i=1}^{k}(\exists\vec{x}(\Delta(IdVar_i, R(\vec{y})|_{y_{\pi_g^{-1}(\upsilon)}=1}\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}) \wedge S(\vec{x}))\{\vec{x}' \leftarrow \vec{x}\})) \vee$$
$$((\overline{\upsilon})\{\vec{x}' \leftarrow \vec{x}\} \cdot (\bigvee_{i=1}^{k}(\exists\vec{x}(\Delta(IdVar_i, R(\vec{y})|_{y_{\pi_g^{-1}(\upsilon)}=0}\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}) \wedge S(\vec{x}))\{\vec{x}' \leftarrow \vec{x}\}))$$

- *Für eine tatsächliche Variable υ zur Kodierung von lokalen Zuständen einer Komponente mit einem Index $i \in M$, lässt sich das relationale Produkt mit einem ETLEBDD aufspalten in:*
  - *υ tatsächliche Variable für aktuelle Zustände ($\upsilon \in X$):*

$$
\begin{aligned}
&(\bigvee_{j \in M \setminus \{i\}} \upsilon \cdot (\exists \vec{x}(\Delta(IdVar'_j, R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_j}\}) \wedge S(\vec{x})|_{\upsilon=1}))\{\vec{x}' \leftarrow \vec{x}\}) \vee \\
&(\bigvee_{j \in M \setminus \{i\}} \overline{\upsilon} \cdot (\exists \vec{x}(\Delta(IdVar'_j, R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_j}\}) \wedge S(\vec{x})|_{\upsilon=0}))\{\vec{x}' \leftarrow \vec{x}\}) \vee \\
&((\exists \vec{x}(\Delta(IdVar, R(\vec{y})|_{y_{\pi_i^{-1}(\upsilon)}=1}\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}) \wedge S(\vec{x})|_{\upsilon=1}))\{\vec{x}' \leftarrow \vec{x}\}) \vee \\
&((\exists \vec{x}(\Delta(IdVar, R(\vec{y})|_{y_{\pi_i^{-1}(\upsilon)}=0}\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}) \wedge S(\vec{x})|_{\upsilon=0}))\{\vec{x}' \leftarrow \vec{x}\})
\end{aligned}
$$

  - *υ tatsächliche Variable für Nachfolgerzustände ($\upsilon \in X'$) und υ' korrespondierende tatsächliche Variable für aktuelle Zustände ($\upsilon \in X$):*

$$
\begin{aligned}
&(\bigvee_{j \in M \setminus \{i\}} \upsilon' \cdot (\exists \vec{x}(\Delta(IdVar'_j, R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_j}\}) \wedge S(\vec{x})|_{\upsilon'=1}))\{\vec{x}' \leftarrow \vec{x}\}) \vee \\
&(\bigvee_{j \in M \setminus \{i\}} \overline{\upsilon'} \cdot (\exists \vec{x}(\Delta(IdVar'_j, R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_j}\}) \wedge S(\vec{x})|_{\upsilon'=0}))\{\vec{x}' \leftarrow \vec{x}\}) \vee \\
&((\upsilon)\{\vec{x}' \leftarrow \vec{x}\} \cdot ((\exists \vec{x}(\Delta(IdVar, R(\vec{y})|_{y_{\pi_i^{-1}(\upsilon)}=1}\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}) \wedge S(\vec{x})))) \\
&\{\vec{x}' \leftarrow \vec{x}\})) \vee ((\overline{\upsilon})\{\vec{x}' \leftarrow \vec{x}\} \cdot ((\exists \vec{x}(\Delta(IdVar, R(\vec{y})|_{y_{\pi_i^{-1}(\upsilon)}=0}\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}) \wedge \\
&S(\vec{x})))\{\vec{x}' \leftarrow \vec{x}\}))
\end{aligned}
$$

Dabei ist $IdVar_j$ jeweils eine Menge von Variablenpaaren, für die für die Komponente mit dem Index j im ETLEBDD implizit Identitätstransformationen berücksichtigt werden. Bei Aufspaltung der Berechnung bezüglich einer tatsächlichen Variable υ zur Kodierung von lokalen Zuständen einer Komponente mit einem Index $i \in M$, sind die Mengen $IdVar_j$ für Komponenten mit Indizes $j \in M$ mit $j \neq i$ nicht leer. Für diese Aufspaltungen sind die Mengen $IdVar'_j$ jeweils die Mengen $IdVar_j$ ohne das Variablenpaar υ, υ', für das die Aufspaltung in Teilprobleme durchgeführt wurde. Das heißt es gilt $IdVar'_j = IdVar_j - \{(\upsilon, \upsilon')\}$ (bzw. $IdVar'_j = IdVar_j - \{(\upsilon', \upsilon)\}$, wenn υ' Variable für aktuelle Zustände und υ korrespondierende Variable für Nachfolgerzustände}).

Beweis. 1. Fall: υ tatsächliche Variable für aktuelle Zustände zur Kodierung des globalen Zustands eines Systems

$$
\begin{aligned}
&\exists\vec{x}[(\Delta(IdVar_1, R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_1}\}) \vee ... \vee \Delta(IdVar_k, R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_k}\}))\\
&\wedge S(\vec{x})]\{\vec{x}' \leftarrow \vec{x}\}\\
\overset{DI}{=}&\exists\vec{x}((\Delta(IdVar_1, R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_1}\}) \wedge S(\vec{x})) \vee ...\\
&\vee (\Delta(IdVar_k, R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_k}\}) \wedge S(\vec{x})))\{\vec{x}' \leftarrow \vec{x}\}\\
\underset{und\ \exists\vec{x}}{\overset{Lemma\ 2.42}{=}}&(\exists\vec{x}(\Delta(IdVar_1, R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_1}\}) \wedge S(\vec{x}))\{\vec{x}' \leftarrow \vec{x}\}) \vee ...\\
&\vee (\exists\vec{x}(\Delta(IdVar_k, R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_k}\}) \wedge S(\vec{x}))\{\vec{x}' \leftarrow \vec{x}\})\\
=&\bigvee_{i=1}^{k}(\exists\vec{x}(\Delta(IdVar_i, R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}) \wedge S(\vec{x}))\{\vec{x}' \leftarrow \vec{x}\})\\
\overset{Satz\ 4.9}{=}&\bigvee_{i=1}^{k}((\exists\vec{x}(\Delta(IdVar_i, R(\vec{y})|_{y_{\pi_i^{-1}(\upsilon)}=1}\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}) \wedge S(\vec{x})|_{\upsilon=1}))\{\vec{x}' \leftarrow \vec{x}\} \vee\\
&(\exists\vec{x}(\Delta(IdVar_i, R(\vec{y})|_{y_{\pi_i^{-1}(\upsilon)}=0}\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}) \wedge S(\vec{x})|_{\upsilon=0}))\{\vec{x}' \leftarrow \vec{x}\})\\
\underset{\pi_i=\pi_g}{\overset{\forall i:}{=}}&(\bigvee_{i=1}^{k}(\exists\vec{x}(\Delta(IdVar_i, R(\vec{y})|_{y_{\pi_g^{-1}(\upsilon)}=1}\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}) \wedge S(\vec{x})|_{\upsilon=1}))\{\vec{x}' \leftarrow \vec{x}\}) \vee\\
&(\bigvee_{i=1}^{k}(\exists\vec{x}(\Delta(IdVar_i, R(\vec{y})|_{y_{\pi_g^{-1}(\upsilon)}=0}\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}) \wedge S(\vec{x})|_{\upsilon=0}))\{\vec{x}' \leftarrow \vec{x}\})
\end{aligned}
$$

2. Fall: υ tatsächliche Variable für Nachfolgerzustände zur Kodierung des globalen Zustands eines Systems

$$
\begin{aligned}
&\exists\vec{x}[(\Delta(IdVar_1, R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_1}\}) \vee ... \vee \Delta(IdVar_k, R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_k}\}))\\
&\wedge S(\vec{x})]\{\vec{x}' \leftarrow \vec{x}\}\\
\overset{DI}{=}&\exists\vec{x}((\Delta(IdVar_1, R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_1}\}) \wedge S(\vec{x})) \vee ...\\
&\vee (\Delta(IdVar_k, R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_k}\}) \wedge S(\vec{x})))\{\vec{x}' \leftarrow \vec{x}\}\\
\underset{und\ \exists\vec{x}}{\overset{Lemma\ 2.42}{=}}&(\exists\vec{x}(\Delta(IdVar_1, R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_1}\}) \wedge S(\vec{x}))\{\vec{x}' \leftarrow \vec{x}\}) \vee ...\\
&\vee (\exists\vec{x}(\Delta(IdVar_k, R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_k}\}) \wedge S(\vec{x}))\{\vec{x}' \leftarrow \vec{x}\})\\
=&\bigvee_{i=1}^{k}(\exists\vec{x}(\Delta(IdVar_i, R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}) \wedge S(\vec{x}))\{\vec{x}' \leftarrow \vec{x}\})
\end{aligned}
$$

$$\overset{Satz\ 4.9}{=} \bigvee_{i=1}^{k}((\upsilon)\{\vec{x}' \leftarrow \vec{x}\} \cdot (\exists\vec{x}(\Delta(IdVar_i, R(\vec{y})|_{y_{\pi_i^{-1}(\upsilon)}=1}\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}) \wedge S(\vec{x})))$$
$$\{\vec{x}' \leftarrow \vec{x}\} \vee (\overline{\upsilon})\{\vec{x}' \leftarrow \vec{x}\} \cdot (\exists\vec{x}(\Delta(IdVar_i, R(\vec{y})|_{y_{\pi_i^{-1}(\upsilon)}=0}\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}) \wedge$$
$$S(\vec{x})))\{\vec{x}' \leftarrow \vec{x}\})$$
$$\overset{\substack{\forall i:\\ \pi_i=\pi_g}}{=} ((\upsilon)\{\vec{x}' \leftarrow \vec{x}\} \cdot (\bigvee_{i=1}^{k}(\exists\vec{x}(\Delta(IdVar_i, R(\vec{y})|_{y_{\pi_g^{-1}(\upsilon)}=1}\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}) \wedge S(\vec{x})))$$
$$\{\vec{x}' \leftarrow \vec{x}\})) \vee ((\overline{\upsilon})\{\vec{x}' \leftarrow \vec{x}\} \cdot (\bigvee_{i=1}^{k}(\exists\vec{x}(\Delta(IdVar_i, R(\vec{y})|_{y_{\pi_g^{-1}(\upsilon)}=0}\{\vec{y} \leftarrow \vec{x}_{\pi_i}\})$$
$$\wedge S(\vec{x})))\{\vec{x}' \leftarrow \vec{x}\}))$$

3. Fall: υ tatsächliche Variable für aktuelle Zustände zur Kodierung von lokalen Zuständen einer Komponente mit einem Index $i \in M$

$$\exists\vec{x}[(\Delta(IdVar_1, R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_1}\}) \vee ... \vee \Delta(IdVar_k, R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_k}\}))$$
$$\wedge S(\vec{x})]\{\vec{x}' \leftarrow \vec{x}\}$$
$$\overset{DI}{=} \exists\vec{x}((\Delta(IdVar_1, R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_1}\}) \wedge S(\vec{x})) \vee ...$$
$$\vee (\Delta(IdVar_k, R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_k}\}) \wedge S(\vec{x})))\{\vec{x}' \leftarrow \vec{x}\}$$
$$\overset{\substack{Lemma\ 2.42\\ und\ \exists\vec{x}}}{=} (\exists\vec{x}(\Delta(IdVar_1, R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_1}\}) \wedge S(\vec{x}))\{\vec{x}' \leftarrow \vec{x}\}) \vee ...$$
$$\vee (\exists\vec{x}(\Delta(IdVar_k, R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_k}\}) \wedge S(\vec{x}))\{\vec{x}' \leftarrow \vec{x}\})$$
$$= \bigvee_{j\in M\setminus\{i\}} (\exists\vec{x}(\Delta(IdVar_j, R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_j}\}) \wedge S(\vec{x}))\{\vec{x}' \leftarrow \vec{x}\}) \vee$$
$$(\exists\vec{x}(\Delta(IdVar_i, R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}) \wedge S(\vec{x}))\{\vec{x}' \leftarrow \vec{x}\})$$
$$\overset{\substack{Sätze\\ 4.8+4.9}}{=} \bigvee_{j\in M\setminus\{i\}} (\upsilon \cdot (\exists\vec{x}(\Delta(IdVar'_j, R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_j}\}) \wedge S(\vec{x})|_{\upsilon=1}))\{\vec{x}' \leftarrow \vec{x}\}) \vee$$
$$\bigvee_{j\in M\setminus\{i\}} (\overline{\upsilon} \cdot (\exists\vec{x}(\Delta(IdVar'_j, R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_j}\}) \wedge S(\vec{x})|_{\upsilon=0}))\{\vec{x}' \leftarrow \vec{x}\}) \vee$$
$$((\exists\vec{x}(\Delta(IdVar, R(\vec{y})|_{y_{\pi_i^{-1}(\upsilon)}=1}\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}) \wedge S(\vec{x})|_{\upsilon=1}))\{\vec{x}' \leftarrow \vec{x}\}) \vee$$
$$((\exists\vec{x}(\Delta(IdVar, R(\vec{y})|_{y_{\pi_i^{-1}(\upsilon)}=0}\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}) \wedge S(\vec{x})|_{\upsilon=0}))\{\vec{x}' \leftarrow \vec{x}\})$$

4. Fall: υ tatsächliche Variable für Nachfolgerzustände zur Kodierung von lokalen Zuständen einer Komponente mit einem Index $i \in M$. υ' ist die zu dieser Variable korrespondierende Variable für aktuelle Zustände

$$
\begin{aligned}
& \exists \vec{x}[(\Delta(IdVar_1, R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_1}\}) \vee ... \vee \Delta(IdVar_k, R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_k}\})) \\
& \wedge S(\vec{x})]\{\vec{x}' \leftarrow \vec{x}\} \\
\overset{DI}{=} & \exists \vec{x}((\Delta(IdVar_1, R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_1}\}) \wedge S(\vec{x})) \vee ... \\
& \vee (\Delta(IdVar_k, R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_k}\}) \wedge S(\vec{x})))\{\vec{x}' \leftarrow \vec{x}\} \\
\overset{\substack{Lemma\ 2.42 \\ und\ \exists \vec{x}}}{=} & (\exists \vec{x}(\Delta(IdVar_1, R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_1}\}) \wedge S(\vec{x}))\{\vec{x}' \leftarrow \vec{x}\}) \vee ... \\
& \vee (\exists \vec{x}(\Delta(IdVar_k, R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_k}\}) \wedge S(\vec{x}))\{\vec{x}' \leftarrow \vec{x}\}) \\
= & \bigvee_{j \in M \setminus \{i\}} (\exists \vec{x}(\Delta(IdVar_j, R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_j}\}) \wedge S(\vec{x}))\{\vec{x}' \leftarrow \vec{x}\}) \vee \\
& (\exists \vec{x}(\Delta(IdVar_i, R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}) \wedge S(\vec{x}))\{\vec{x}' \leftarrow \vec{x}\}) \\
\overset{\substack{Sätze \\ 4.8+4.9}}{=} & \bigvee_{j \in M \setminus \{i\}} (\upsilon' \cdot (\exists \vec{x}(\Delta(IdVar'_j, R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_j}\}) \wedge S(\vec{x})|_{\upsilon'=1}))\{\vec{x}' \leftarrow \vec{x}\}) \vee \\
& \bigvee_{j \in M \setminus \{i\}} (\overline{\upsilon'} \cdot (\exists \vec{x}(\Delta(IdVar'_j, R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_j}\}) \wedge S(\vec{x})|_{\upsilon'=0}))\{\vec{x}' \leftarrow \vec{x}\}) \vee \\
& ((\upsilon)\{\vec{x}' \leftarrow \vec{x}\} \cdot ((\exists \vec{x}(\Delta(IdVar, R(\vec{y})|_{y_{\pi_i^{-1}(\upsilon)}=1}\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}) \wedge S(\vec{x}))) \\
& \{\vec{x}' \leftarrow \vec{x}\})) \vee ((\overline{\upsilon})\{\vec{x}' \leftarrow \vec{x}\} \cdot ((\exists \vec{x}(\Delta(IdVar, R(\vec{y})|_{y_{\pi_i^{-1}(\upsilon)}=0}\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}) \wedge \\
& S(\vec{x})))\{\vec{x}' \leftarrow \vec{x}\}))
\end{aligned}
$$

□

Der Algorithmus *RelProductETLEBDD()* führt die rekursive Berechnung des relationalen Produkts beginnend mit dem Wurzelknoten des BDDs des ETLEBDDs zur Repräsentation einer Menge von Transitionen und dem Wurzelknoten des BDDs zur Repräsentation einer Zustandsmenge für Komponenten auf Berechnungspfaden gemeinsam durch, bis sich die für Komponenten durchzuführenden Berechnungen in einem Rekursionsschritt unterscheiden. Sind in einem Rekursionsschritt für die noch zu betrachtenden Komponenten die gleichen Berechnungen auszuführen, erfolgt keine Aufsplittung der Berechnung in für Komponenten unterschiedliche Berechnungen und der Algorithmus *RelProductETLEBDD()* ruft zur weiteren rekursiven Berechnung die in Algorithmus 15 angegebene Funktion *StraightExecution()* auf. Dies erfolgt (siehe Zeile 21 im Algorithmus *RelProductETLEBDD()*), wenn sich der Algorithmus bereits in seinem Ausführungsmodus *iso* befindet, die Top-Variable des Rekursionsschritts eine Variable zur Kodierung von globalen Zuständen ist oder wenn die Top-Variable eine Variable zur Kodierung von lokalen Zuständen einer Komponente ist, für die auf dem aktuellen Berechnungspfad bereits vorher in den Ausführungsmodus *iso* verzweigt wurde (previousSplitComp[CompLocVar(υ)]==TRUE)).

Ist die Top-Variable eines Rekursionsschritts eine Variable zur Kodierung von globalen Zuständen, so wird die weitere Berechnung im Algorithmus im Rekursionsschritt immer durch Aufruf der Funktion *StraightExecution()* durchgeführt. Die ersten beiden in Satz 5.7 angegebenen Aufspaltungen in Teilprobleme sind für solche Top-Variablen gültig. Es sind bei diesen Aufspaltungen in Teilprobleme für alle noch zu betrachtenden Komponenten die gleichen BDDs der Kofaktoren R und S in den Teilproblemen vorhanden ($R(\vec{y})|_{\pi_g^{-1}(\upsilon)=1}$ ($R(\vec{y})|_{\pi_i^{-1}(\upsilon)=1}$, da $\pi_g == \pi_i$) bzw. $R(\vec{y})|_{\pi_g^{-1}(\upsilon)=0}$ ($R(\vec{y})|_{\pi_i^{-1}(\upsilon)=0}$, da $\pi_g == \pi_i$) und $S(\vec{x})|_{\upsilon=1}$, $S(\vec{x})|_{\upsilon=0}$ bzw. $S(\vec{x})$). Diese Aufspaltungen entsprechen den in der Funktion *StraightExecution()* verwendeten Aufspaltungen in Teilprobleme und die Kofaktoren den bei den rekursiven Aufrufen verwendeten Kofaktoren (siehe Zeilen 7 und 10 bzw. Zeilen 15 und 18 in der Funktion *StraightExecution()*). Bei einer Top-Variablen für Nachfolgerzustände υ gleichen die in der Funktion verwendeten Kofaktoren $S(\vec{x})|_{\upsilon=1}$ bzw. $S(\vec{x})|_{\upsilon=0}$ dem Kofaktor $S(\vec{x})$ in der für diesen Fall in Satz 5.7 angegebenen zweiten Aufspaltung in Teilprobleme, da im BDD S nur Variablen für aktuelle Zustände wesentliche Variablen sind. Wird die Funktion *StraightExecution()* bei einer Top-Variablen zur Kodierung von lokalen Zuständen einer Komponente aufgerufen, sind zwei verschiedene Fälle möglich. Der Algorithmus kann gerade nur noch für eine Komponente gültige Berechnungen in seinem Ausführungsmodus *iso* ausführen oder die Top-Variable kann eine lokale Variable einer Komponente, für die auf dem aktuellen Berechnungspfad bereits vor dem aktuellen Rekursionsschritt in den Ausführungsmodus *iso* verzweigt wurde, sein. In der Funktion *StraightExecution()* wird hier die dritte und vierte in Satz 5.7 als korrekt bewiesene Aufspaltung in Teilprobleme angewandt.

In der dritten und vierten in Satz 5.7 angegebenen Aufspaltung in Teilprobleme sind jeweils zwei verschiedene Arten von Teilproblemen vorhanden. Zuerst sind dort Teilprobleme bezüglich einer Belegung der Variable υ mit den beiden Booleschen Werten angegeben, die für Komponenten für die die Variable υ eine Variable zur Kodierung einer Identitätstransformation ist, gültig sind. Die weiteren beiden angegebenen Aufspaltungen sind für eine Komponente i gültig, für die die Top-Variable υ eine Variable zur Kodierung ihres lokalen Zustands ist. Ist die Top-Variable υ für eine Komponente eine Variable zur Kodierung von deren lokalen Zuständen, dann ist diese für die anderen im System vorhandenen Komponenten eine Variable zur Kodierung einer Identitätstransformation. Wird die Funktion *StraightExecution()* bei einer Top-Variable zur Kodierung des lokalen Zustands einer Komponente, die nicht in der Menge der Komponenten für die auf dem aktuellen Berechnungspfad noch Berechnungen erfolgen enthalten ist aufgerufen, ist die Top-Variable für die noch zu betrachtenden Komponenten eine Variable zur Kodierung einer Identitätstransformation. Daher sind von der dritten und vierten im Satz für solche Variablen angegebenen Aufspaltung in Teilprobleme nur die ersten beiden im Satz für die Komponentenmenge $\bigvee_{j \in M \setminus \{i\}}$ angegebenen Teilberechnungen durchzuführen. Diese enthalten die für alle noch zu berücksichtigenden Komponenten auszuführenden Berechnungen. Es ist zu beachten, dass im Algorithmus *RelProductETLEBDD()* $\upsilon \neq$ *mappedVariable* gilt, da eine dem Wurzelknoten des BDDs des ETLEBDDs zugeordnete tatsächliche Variable hier einer später in der Variablenordnung vorkommenden Variablen entspricht. Damit gilt für die Kofaktoren $R_{w_1} = R_{w_0} = R$. Die gleichen Aufspaltungen werden vom Algorithmus

in der Funktion *StraightExecution()* verwendet, wenn er im selben Fall im Ausführungsmodus *iso* nur noch für eine Komponente gültige Berechnungen durchführt.

Im Ausführungsmodus *iso* des Algorithmus kann es auch sein, dass die Top-Variable eine Variable zur Kodierung von lokalen Zuständen der Komponente, für die isoliert Berechnungen ausgeführt werden, ist. Dafür geben jeweils die letzten beiden Teilprobleme in den in Satz 5.7 für eine Variable zur Kodierung von lokalen Zuständen einer Komponente i als Top-Variable angegebenen Aufspaltungen gültige Aufspaltungen an. Die in den beschriebenen Aufspaltungen und den genannten Teilproblemen verwendeten Kofaktoren entsprechen den Kofaktoren, die in den entsprechenden Fällen auch in der Funktion *StraightExecution()* bei rekursiven Aufrufen zur Lösung von Teilproblemen benutzt werden. Mit den damit rekursiv ermittelten Teilergebnissen sind gemäß der in Satz 5.7 bewiesenen Aufspaltungen drei verschiedene Arten zur Kombination zum Rückgabewert eines Rekursionsschritts durchzuführen. Dies sind entweder die Veroderung der Teilergebnisse, die Veroderung und Kombination mit einem bestimmten Wert für die umbenannte aktuelle Top-Variable ("$(\upsilon)\{\vec{x}' \leftarrow \vec{x}\}\ \cdot$", entspricht im Resultats-BDD dem Erzeugen eines neuen Knotens für die durch die Umbenennung erhaltene Variable mit den rekursiv ermittelten Teilergebnissen als Söhnen) oder nur die Kombination mit einem bestimmten Wert für die aktuelle Top-Variable ("$\upsilon\ \cdot$"). In der Funktion *StraightExecution()* werden die Teilresultate durch die Funktion *ResultCalculation()* auf diese Weise wie in den Aufspaltungen in Teilprobleme angegeben kombiniert. Damit wird bei Benutzung der Funktion *StraightExecution()* genau wie in den in Satz 5.7 angegebenen Aufspaltungen verfahren.

Die rekursive Berechnung und die Aufspaltung in Teilprobleme erfolgt im Algorithmus *RelProductETLEBDD()* solange für mehrere Komponenten gemeinsam, bis eine Variable zur Kodierung des lokalen Zustands einer dieser Komponenten Top-Variable eines Rekursionsschritts ist und für diese Komponente auf dem aktuellen Berechnungspfad noch keine Aufspaltung in den Ausführungsmodus *iso* durchgeführt wurde. In einem solchen Rekursionsschritt unterscheiden sich die für diese Komponente und die für die restlichen Komponenten durchzuführenden Berechnungen. Daher führt der in Abschnitt 5.3 eingeführte Algorithmus dort eine Aufspaltung der Berechnung in nur für die eine Komponente gültige Berechnungen und in für die restlichen noch zu betrachtenden Komponenten gültige Berechnungen durch. Dies wird durch Aufruf der Funktion *ExecutionSplit()* (siehe Zeile 25 im Algorithmus *RelProductETLEBDD()*) ausgeführt. In dieser erfolgt eine Aufspaltung der Berechnung in Teilprobleme für die Komponente, für die die Top-Variable eine Variable zur Kodierung von lokalen Zuständen ist und in Teilprobleme für die restlichen Komponenten für die noch Berechnungen durchgeführt werden. Bei Betrachtung der bei den rekursiven Aufrufen des Algorithmus *RelProductETLEBDD()* in der Funktion *ExecutionSplit()* hier verwendeten Kofaktoren sieht man, dass diese ebenfalls den in Satz 5.7 bei den Aufspaltungen für Variablen zur Kodierung von lokalen Zuständen verwendeten Kofaktoren entsprechen. Für die Komponente, für die die Top-Variable des aktuellen Rekursionsschritts eine Variable zur Kodierung von lokalen Zuständen ist, geben die letzten beiden Teilprobleme in den dafür in Satz 5.7 angegebenen Aufspaltungen in Teilprobleme gültige Aufspaltungen an. Bei diesen werden Kofaktoren des BDDs R bezüglich der der Top-Variablen entsprechenden reduzierten Variablen verwendet $(R(\vec{y})|_{\pi_i^{-1}(\upsilon)=1}$ bzw.

$R(\vec{y})|_{\pi_i^{-1}(\upsilon)=0}$). Die bei den rekursiven Aufrufen des Algorithmus *RelProductETLEBDD()* für diese Komponente in der Funktion *ExecutionSplit()* verwendeten Kofaktoren R_{w_1} bzw. R_{w_0} (siehe Zeilen 8 und 10 bzw. 13 und 15) entsprechen diesen Kofaktoren. Die Kofaktoren des BDDs S werden direkt wie in diesen Aufspaltungen in Satz 5.7 angegeben verwendet (für eine Top-Variable υ für aktuelle Zustände: $S(\vec{x})|_{\upsilon=1}$ bzw. $S(\vec{x})|_{\upsilon=0}$; für eine Top-Variable υ für Nachfolgerzustände: $S(\vec{x})$). Bei den für die restlichen noch zu betrachtenden Komponenten gültigen Aufspaltungen in Teilprobleme wird der BDD $R(\vec{y})$ in diesem Fall im Satz unverändert verwendet. Ebenso wird dieser in der Funktion *ExecutionSplit()* bei den hierfür durchgeführten rekursiven Aufrufen des Algorithmus als Übergabeparameter benutzt (siehe Zeilen 29 und 31). Für diese Komponenten, für die die Top-Variable in diesem Fall eine Variable zur Kodierung eines Identitätsmusters ist, werden die rekursiven Aufrufe im Algorithmus für beide korrespondierenden Variablen eines Identitätsmusters gemeinsam ausgeführt. Dazu werden sowohl bei den im Satz dafür bewiesenen Aufspaltungen, als auch im Algorithmus jeweils Kofaktoren des BDDs S bezüglich der Variablen für aktuelle Zustände des korrespondierenden Variablenpaares verwendet (siehe Zeilen 24 und 25 in der Funktion *ExecutionSplit()*, in der die korrespondierende Variable für aktuelle Zustände falls nötig ermittelt wird). Die Resultatskombination erfolgt im Algorithmus, für die nur für eine Komponente gültigen Berechnungen (siehe Zeile 17) und bei den für mehrere Komponenten gemeinsam durchzuführenden Berechnungen (siehe Zeile 33), jeweils zuerst separat durch Aufruf der Funktion *ResultCalculation()*. Dort werden die zuvor bereits für die Funktion *StraightExecution()* beschriebenen Operationen, die für die Ermittlung der Teilergebnisse in Satz 5.7 angegeben sind, ausgeführt. Durch Veroderung dieser beiden Ergebnisse wird das gesamte Ergebnis eines Rekursionsschritts danach ebenfalls wie es im Satz angegeben ist berechnet (siehe Zeile 34). Damit entspricht das Vorgehen des Algorithmus *RelProductETLEBDD()* bei der rekursiven Aufspaltung in Teilprobleme den in Satz 5.7 als korrekt bewiesenen Aufspaltungen.

5.7 Korrektheit der Terminalfälle

Die im Algorithmus *RelProductETLEBDD()* verwendeten Terminalfälle und deren Rückgabewerte sind in Algorithmus 17 angegeben. Deren Korrektheit wird im Folgenden erläutert. In einem ETLEBDD kann eine Menge isomorpher Transitionen, die Transitionen mehrerer Komponenten beinhaltet, gespeichert werden. Der in Abschnitt 5.3 beschriebene Algorithmus ermöglicht die Berechnung des relationalen Produkts mit allen mit einem ETLEBDD von Komponenten repräsentierten Transitionen auf einmal. Außerdem nutzt der Algorithmus mit dem Wurzelknoten des ETLEBDDs bzw. dem des beteiligten BDDs beginnend für mehrere Komponenten auf Berechnungspfaden gleich anfallende Teilberechnungen aus, indem er diese nur einmal ausführt. Bei der Berechnung des relationalen Produkts zwischen einem TLEBDD und einem BDD ergibt sich der für eine Eingabebelegung resultierende Boolesche Wert des relationalen Produkts aus der UND-Verknüpfung der Booleschen Werte der durch die Eingabebelegung erreichten Terminalknoten. Dabei sind noch die in einem TLEBDD fehlenden Knoten für Identitätstransformationen zur Kodierung der Änderungen von lokalen Zuständen von Komponenten, für die ein TLEBDD

nicht gebaut wurde, zu berücksichtigen (siehe Definition TLEBDD in Abschnitt 4.1). Der Algorithmus *RelProductETLEBDD()* berechnet das relationale Produkt für Transitionen mehrerer Komponenten gemeinsam. Daher muss der Rückgabewert eines in einem Rekursionsschritt des Algorithmus eingetretenen Terminalfalls der Vereinigung des sich hier bei einer separaten Berechnung für die einzelnen Komponenten ergebenden Resultats entsprechen. Bei der separaten weiteren Berechnung für eine Komponente wird der BDDs des ETLEBDDs nur für diese Komponente berücksichtigt. ETLEBDDs sind eine Erweiterung von TLEBDDs zur Speicherung von Zuordnungslisten zur Zuordnung von reduzierten Variablen auf die diesen entsprechenden tatsächlichen Variablen für mehrere Komponenten. Sie wurden zur gemeinsamen Speicherung von Transitionen mehrerer Komponenten entworfen. Wird der BDD des ETLEBDDs nur für eine Komponente berücksichtigt, wird zur Abbildung der reduzierten Variablen dieses BDDs auf die korrespondierenden tatsächlichen Variablen ausschließlich die Zuordnungsliste dieser Komponente verwendet. Dies entspricht der Berücksichtigung der Zuordnungsinformationen auf die tatsächlichen Variablen von nur einer Komponente bei TLEBDDs. Damit gleicht die hier für eine Komponente durchzuführende Berechnung der beim relationalen Produkt mit einem TLEBDD für diese Komponente für das gegebene Teilproblem durchzuführenden Berechnung. Aus diesem Grund kann die Korrektheit der im Algorithmus *RelProductETLEBDD()* verwendeten Terminalfälle direkt aus der Korrektheit der beim Algorithmus zur Berechnung des relationalen Produkts zwischen einem TLEBDD und einem BDD verwendeten Terminalfälle (siehe Abschnitt 4.5) abgeleitet werden.

Beim Algorithmus zur Berechnung des relationalen Produkts mit einem TLEBDD liegen Terminalfälle mit dem Terminalknoten für den Booleschen Wert 0 als Rückgabewert vor, wenn der aktuelle Wurzelknoten des BDDs des TLEBDDs oder der aktuelle Wurzelknoten des BDDs zur Repräsentation der zu explorierenden Zustandsmenge in einem Rekursionsschritt ein Terminalknoten für den Booleschen Wert 0 ist. Ist der BDD R des ETLEBDDs oder der BDD S in einem Rekursionsschritt des Algorithmus für ETLEBDDs ein Terminalknoten für den Booleschen Wert 0, dann ergibt sich bei der separaten Berechnung des relationalen Produkts für alle noch zu berücksichtigenden Komponenten mit dem Algorithmus für TLEBDDs der Boolesche Wert 0 als Rückgabewert. Durch die Vereinigung dieses Rückgabewerts für alle Komponenten ergibt sich für das relationale Produkt mit einem ETLEBDD ebenfalls der Rückgabewert 0. Ein weiterer beim relationalen Produkt mit einem ETLEBDD verwendeter Terminalfall tritt ein, wenn der Wurzelknoten des BDDs R des ETLEBDDs und der Wurzelknoten des BDDs S beide Terminalknoten für den Booleschen Wert 1 sind. Für diesen Terminalfall wurde in Abschnitt 4.5 der Terminalknoten für den Booleschen Wert 1 als korrekter Rückgabewert bewiesen. Tritt dieser Terminalfall beim Algorithmus zur Berechnung des relationalen Produkts mit einem ETLEBDD auf, ergibt sich bei separater Fortführung der Berechnung für jede zu berücksichtigende Komponente der Rückgabewert 1. Der Terminalknoten für den Booleschen Wert 1 ist damit auch beim Algorithmus für ETLEBDDs der korrekte Rückgabewert eines solchen Rekursionsschritts.

In der im Algorithmus *RelProductETLEBDD()* verwendeten Funktion *TerminalCase()* sind zwei weitere Terminalfälle aufgeführt, die auftreten, wenn der Wurzelknoten des

BDDs R des ETLEBDDs der Terminalknoten für den Booleschen Wert 1 ist. Beide Terminalfälle wurden auch im Algorithmus zur Berechnung des relationalen Produkts zwischen einem TLEBDD und einem BDD in ähnlicher Form verwendet. Einer dieser Terminalfälle tritt beim für ETLEBDDs vorgestellten Algorithmus ein, wenn zusätzlich in der Variablenordnung ab der Variablen des Wurzelknotens des BDDs S nur noch Variablen zur Kodierung des globalen Zustands oder des lokalen Zustands der Komponente, auf deren Index die Variable *compIndex* gerade zeigt, vorkommen (Zeile 6 in Algorithmus 17). Im Algorithmus zur kombinierten Berechnung des relationalen Produkts mit einem TLEBDD wird für einen Rekursionsschritt mit diesen Eigenschaften ebenfalls ein Terminalfall, der den Terminalknoten für den Booleschen Wert 1 als Rückgabewert besitzt, verwendet. Da die im für ETLEBDDs präsentierten Algorithmus für die Komponente auf die *compIndex* gerade zeigt durchzuführende weitere Berechnung dieser Berechnung im Algorithmus für TLEBDDs entspricht, ist das Teilergebnis des Algorithmus für einen ETLEBDD für diese Komponente der Terminalknoten für den Booleschen Wert 1. Der Rückgabewert eines Rekursionsschritts im Algorithmus für ETLEBDDs ist die Vereinigung der für alle noch zu berücksichtigenden Komponenten resultierenden Teilergebnisse. Da das Teilergebnis der Komponente auf die *compIndex* zeigt der Terminalknoten für den Booleschen Wert 1 ist, ist das Ergebnis der Vereinigung ebenfalls dieser Terminalknoten und der Terminalknoten ist auch im Algorithmus für ETLEBDDs der korrekte Rückgabewert für den beschriebenen Terminalfall. Der letzte verwendete Terminalfall tritt ein, wenn der Wurzelknoten des BDDs R des ETLEBDDs ein Terminalknoten für den Booleschen Wert 1 ist, nur noch Berechnungen für eine Komponente durchzuführen sind (Algorithmus ist in seinem Ausführungsmodus *iso*) und in der Variablenordnung ab der Variablen des Wurzelknotens des BDDs S für diese Komponente nur noch Variablen zur Kodierung von Identitätstransformationen, die für andere Komponenten Variablen zur Kodierung des lokalen Zustands sind, vorkommen (Zeile 9). Da in diesem Rekursionsschritt nur noch Berechnungen für eine Komponente auszuführen sind, entspricht dieser Terminalfall direkt dem Terminalfall im Algorithmus zur Berechnung des relationalen Produkts mit einem TLEBDD, in dem für die Komponente, für die der TLEBDD gebaut wurde, ebenfalls nur noch Variablen zur Kodierung solcher Identitätstransformationen vorkommen. Damit ist der dafür als korrekt bewiesene Rückgabewert des Terminfalfalls, der BDD S, auch im Algorithmus für ETLEBDDs der korrekte Rückgabewert.

5.8 Erreichbarkeitsanalyse mit ETLEBDDs

Bei der Erreichbarkeitsanalyse mit Verwendung von ETLEBDDs zur Repräsentation der Transitionsrelation können verschiedene Strategien zur Exploration des Zustandsraums angewandt werden. Im Folgenden werden Explorationsstrategien für die Erreichbarkeitsanalyse für asynchrone nebenläufige Systeme mit Transitionslokalität aus M Komponenten ($M > 0$) vorgestellt. Dabei wird von einer Repräsentation der Transitionsrelation durch ETLEBDDs und TLEBDDs ausgegangen. Mit dem ETLEBDD wird eine Menge isomorpher Transitionen repräsentiert. Die restlichen Transitionen werden in TLEBDDs gespeichert. Die Explorationsstrategien geben Ausführungsreihenfolgen von Bildberechnun-

gen mit ETLEBDDs und TLEBDDs an. Da die Explorationsstrategien für die Vorwärts- und Rückwärtserreichbarkeitsanalyse (siehe Abschnitt 2.1.6.3) verwendet werden können, werden diese nachfolgend allgemein für die Durchführung von Bildberechnungen, die Vorgänger- oder Nachfolgerberechnungen sein können, vorgestellt. Dabei wird von einer komponentenweisen Partitionierung (siehe Abschnitt 2.5.2) der mit TLEBDDs repräsentierten Teile der Transitionsrelation, aus einer Partition und damit einem TLEBDD für jede im System vorhandene Komponente (M Partitionen), ausgegangen. Sollen Transitionen einer Komponente durch mehrere TLEBDDs repräsentiert werden, muss in den nachfolgend vorgestellten Explorationsstrategien die über die M TLEBDDs iterierende Schleife entsprechend auf eine größere Anzahl an TLEBDDs erweitert werden.

In den angegebenen Explorationsstrategien sind Bildberechnungen, bei denen das relationale Produkt berechnet wird, für TLEBDDs durch Aufruf der Funktion $Image_{TLE}()$ dargestellt. Bei der Vorwärtserreichbarkeitsanalyse könnten die dort verwendeten Nachfolgerberechnungen zum Beispiel mit dem in Abschnitt 4.3 vorgestellten Algorithmus zur kombinierten Berechnung des relationalen Produkts mit einem TLEBDD durchgeführt werden. Bildberechnungen mit ETLEBDDs sind durch Aufruf der Funktion $Image_{ETLE}()$ angegeben. Für eine Nachfolgerberechnung könnte dafür zum Beispiel der in Abschnitt 5.3 präsentierte Algorithmus zur kombinierten Berechnung des relationalen Produkts mit einem ETLEBDD benutzt werden.

Im Folgenden werden Explorationsstrategien durch Angabe von Vorgehensweisen zur Durchführung von Bildberechnungen mit allen Transitionen der Transitionsrelation angegeben. Eine Anwendung der präsentierten Vorgehensweisen aus Bildberechnungen mit allen Transitionen wird nachfolgend als Explorationsrunde bezeichnet. Bei der Erreichbarkeitsanalyse werden wiederholt solche Explorationsrunden aus Bildberechnungen ausgeführt, bis alle erreichbaren Zustände gefunden wurden. Für das Vorgehen in einer Explorationsrunde werden hier nur die für die Durchführung der Bildberechnungen wichtigen Teile angegeben. Um die Vorgehensweisen für die Erreichbarkeitsanalyse verwenden zu können, sind gewöhnlich noch weitere Berechnungen (wie zum Beispiel die Anpassung der Menge *Reached* mit allen bisher gefundenen Zuständen nach jeder Explorationsrunde) auszuführen. Dies kann ähnlich wie in Abschnitt 2.1.6.3 für die Vorwärtserreichbarkeitsanalyse vorgestellt erfolgen, wobei die Menge S_{k+1} bei den in diesem Abschnitt nachfolgend vorgestellten Strategien der Menge der in der nächsten Explorationsrunde zu explorierenden Zustände (*ToExplore* in Abschnitt 2.1.6.3) entspricht.

Algorithmus 20: Eine Explorationsrunde mit einer ausschließlich durch TLEBDDs repräsentierten komponentenweise partitionierten Transitionsrelation. Es wird eine Bildberechnung mit jedem TLEBDD durchgeführt.

```
1 foreach i ∈ M do
2     S_{k+1} = S_{k+1} ∪ (Image_{TLE}(R_i, S_k) \ S_{Reached});
```

In Algorithmus 20 wird ein Ansatz vorgestellt, bei dem bei Verwendung einer partitionierten Transitionsrelation in einer Explorationsrunde eine Bildberechnung mit jeder

Partition der Transitionsrelation erfolgt. Dieser kann sowohl für Repräsentationen der Transitionsrelation durch TLEBDDs, als auch durch BDDs, verwendet werden. Jede Bildberechnung wird mit einer Partition der Transitionsrelation R_i und einer Menge an weiter zu explorierenden Zuständen S_k durchgeführt. Von den durch eine Bildberechnung ermittelten Zuständen werden nach deren Berechnung die während der Erreichbarkeitsanalyse in vorherigen Explorationsrunden bereits gefundenen Zustände $S_{Reached}$ abgezogen ($Image_{TLE}(R_i, S_k) \setminus S_{Reached}$). Anschließend werden die in vorherigen Explorationsrunden noch nicht ermittelten Zustände mit den anderen in der aktuellen Explorationsrunde bereits neu gefundenen Zuständen S_{k+1} vereinigt.

Im Folgenden werden zwei Strategien zur Erreichbarkeitsanalyse bei Verwendung von ETLEBDDs vorgestellt. Für beide werden in Kapitel 8 experimentelle Ergebnisse präsentiert. Mit einem ETLEBDD kann eine Menge isomorpher Transitionen, die Transitionen verschiedener Komponenten beinhaltet, repräsentiert werden. Transitionen, die nicht in einer mit einem ETLEBDD repräsentierten Menge isomorpher Transitionen enthalten sind, werden in den hier für ETLEBDDs präsentierten Ansätzen weiterhin komponentenweise durch separate TLEBDDs repräsentiert. Die erste Explorationsstrategie für die Verwendung von ETLEBDDs ist in Algorithmus 21 angegeben. In einer Explorationsrunde wird dort zuerst eine Bildberechnung mit der Menge an weiter zu explorierenden Zuständen S_k und einem ETLEBDD durchgeführt (Zeile 1 in Algorithmus 21). Anschließend werden Bildberechnungen mit jeder durch einen TLEBDD repräsentierten Partition durchgeführt (Zeile 3 in Algorithmus 21).

Algorithmus 21: Eine Explorationsrunde bei Benutzung von ETLEBDDs und der Explorationsstrategie *einfache simultane Exploration*.

```
1 S_{k+1} = S_{k+1} ∪ (Image_{ETLE}(R_{ETLEBDD}, S_k) \ S_{Reached});
2 foreach i ∈ M do
3     S_{k+1} = S_{k+1} ∪ (Image_{TLE}(R_i, S_k) \ S_{Reached});
```

Mit dem in Abschnitt 5.3 für die Nachfolgerberechnung mit ETLEBDDs vorgestellten Algorithmus können Nachfolgerzustände mehrerer Komponenten auf einmal berechnet werden. Dabei kann der Algorithmus für mehrere Komponenten bei der Nachfolgerberechnung gleich anfallende Teilberechnungen ausnutzen, in dem diese nur einmal ausgeführt werden. Durch die gemeinsame Nachfolgerberechnung für Transitionen mehrerer Komponenten können mit diesem Algorithmus bei Nachfolgerberechnungen viele Nachfolgerzustände auf einmal exploriert werden. Dies wird in der zweiten in diesem Abschnitt für die Benutzung von ETLEBDDs vorgestellten Explorationsstrategie, die in Algorithmus 22 angegeben ist, ausgenutzt. Bei dieser werden in einer Explorationsrunde zuerst solange Bildberechnungen mit dem ETLEBDD ausgeführt, bis damit keine neuen Zustände mehr gefunden werden können. Dazu wird bei diesem Ansatz eine *while*-Schleife verwendet, in der in jeder Iteration eine Bildberechnung mit dem ETLEBDD und einer Menge an aktuell weiter zu explorierenden Zuständen erfolgt (Zeile 3 in Algorithmus 22). Die durch die Bildberechnung neu gefundenen Zustände werden anschließend zur Menge $S_{NewSimul}$ hin-

zugefügt (Zeile 4). In dieser werden alle innerhalb einer Ausführung der *while*-Schleife neu gefundenen Zustände gespeichert. Falls in einer Iteration der Schleife bisher noch nicht entdeckte Zustände gefunden wurden ($S_{Simul} \neq \emptyset$), wird in der nächsten Iteration eine Bildberechnung mit diesen Zuständen durchgeführt. Können durch Bildberechnungen mit dem ETLEBDD keine noch unentdeckten Zustände mehr gefunden werden, wird die Menge der vor dieser Explorationsrunde unexplorierten Zustände S_k mit der Menge der durch Bildberechnungen mit dem ETLEBDD neu gefundenen Zustände vereinigt (siehe Zeile 6 in Algorithmus 22). Anschließend werden Bildberechnungen mit den TLEBDDs und der neu berechneten Zustandsmenge S_k durchgeführt. Dadurch werden die durch Bildberechnungen mit dem ETLEBDD neu gefundenen Zustände direkt weiterexploriert und die Bildberechnungen mit den TLEBDDs können für größere Zustandsmengen ausgeführt werden. Nach Ausführung einer in Algorithmus 22 angegebenen Explorationsrunde, wurden die Zustände der in Zeile 6 ermittelten Menge S_k mit den im ETLEBDD enthaltenen Transitionen und mit den durch TLEBDDs repräsentierten Transitionen weiterexploriert. Deshalb müssen diese Zustände nicht zur Menge S_{k+1}, die nach einer Explorationsrunde alle in dieser neu gefundenen Zustände enthält und die die Basis für die bei den nächsten Bildberechnungen zu explorierenden Zustände bildet, hinzugefügt werden. Ist die gesamte Transitionsrelation eines Systems eine Menge isomorpher Transitionen, kann die gesamte Transitionsrelation in einem ETLEBDD gespeichert werden und es müssen nur Bildberechnungen mit ETLEBDDs durchgeführt werden. Erfolgt die Repräsentation der Transitionen ausschließlich durch einen ETLEBDD, werden in beiden für ETLEBDDs vorgestellten Explorationsstrategien durch die Bildberechnungen mit dem ETLEBDD alle erreichbaren Zustände gefunden. Beim Verfahren aus Algorithmus 22 können hier in einer Ausführung der *while*-Schleife alle erreichbaren Zustände direkt gefunden werden.

Algorithmus 22: Eine Explorationsrunde bei Benutzung von ETLEBDDs und der Explorationsstrategie *mehrfache simultane Exploration*.

```
1 S_Simul = S_k;
2 while S_Simul ≠ ∅ do
3 |   S_Simul = Image_ETLE(R_ETLEBDD, S_Simul) \ S_Reached;
4 |   S_NewSimul = S_NewSimul ∪ S_Simul;
5 |   S_Reached = S_Reached ∪ S_Simul;
6 S_k = S_k ∪ S_NewSimul;
7 foreach i ∈ M do
8 |   S_{k+1} = S_{k+1} ∪ (Image_TLE(R_i, S_k) \ S_Reached);
```

5.9 Abgrenzung zu verwandten Arbeiten

In diesem Kapitel wurden die Datenstruktur ETLEBDD zur Repräsentation einer Menge isomorpher Transitionen, ein Algorithmus zur simultanen Berechnung des relationalen Produkts mit einem ETLEBDD für mehrere Komponenten und Explorationsstrategien

zur Erreichbarkeitsanalyse mit ETLEBDDs vorgestellt. Diese Ansätze werden in diesem Abschnitt mit verwandten anderen Arbeiten, von denen die meisten in Abschnitt 2.5 der vorliegenden Arbeit vorgestellt wurden, verglichen. Eine Abgrenzung der in Kapitel 4 eingeführten TLEBDDs zu anderen Ansätzen zur effizienten Repräsentation der Transitionsrelation wurde in Abschnitt 4.6 dieser Arbeit durchgeführt. In einem TLEBDD können Transitionen einer Komponente eines asynchronen nebenläufigen Systems mit Transitionslokalität gespeichert werden. Ein ETLEBDD, der wie ein TLEBDD aus einem BDD über einer reduzierten Variablenmenge und Zuordnungsinformationen zur Abbildung der reduzierten Variablen auf die entsprechenden tatsächlichen Variablen besteht, ist eine Erweiterung eines TLEBDDs zur gemeinsamen Repräsentation von Transitionen mehrerer Komponenten. ETLEBDDs unterscheiden sich von TLEBDDs dadurch, dass bei einem ETLEBDD zur Speicherung der Zuordnungsinformationen mehrerer Komponenten ein Array aus Zuordnungslisten verwendet wird. Durch die Speicherung der Zuordnungslisten unterschiedlicher Komponenten in verschiedenen Arrayeinträgen, können die reduzierten Variablen des BDDs des ETLEBDDs für mehrere Komponenten den diesen entsprechenden tatsächlichen Variablen zugeordnet werden. Ein TLEBDD besitzt hingegen nur eine einzelne Zuordnungsliste für eine Komponente. Die Menge der reduzierten Variablen besteht bei Betrachtung dieser für eine Komponente eines Systems auch bei einem ETLEBDD aus Booleschen Variablen zur Kodierung des globalen Zustands des Systems und aus Booleschen Variablen zur Kodierung von lokalen Zuständen dieser Komponente. Bei Interpretation der im BDD des ETLEBDDs gespeicherten Zustandsübergänge für eine Komponente wird, wie in einem TLEBDD, für korrespondierende tatsächliche Variablen zur Kodierung der Änderungen des lokalen Zustands aller anderen Komponenten implizit die Interpretation von Identitätstransformationen angenommen. Im Gegensatz zu einem gewöhnlichen BDD werden solche Identitätstransformationen wie bei TLEBDDs auch bei einem ETLEBDD nicht explizit repräsentiert und es sind für diese Identitätstransformationen daher keine Knoten im BDD des ETLEBDDs vorhanden. Deshalb besitzt dieser ebenfalls nur eine reduzierte Menge an Eingabevariablen. Damit ist die in Abschnitt 4.6 für TLEBDDs durchgeführte Abgrenzung bezüglich anderer Verfahren, bei denen eine Umbenennung der Eingabevariablen durchgeführt wird oder bei denen gewisse Identitätstransformationen nicht explizit abgespeichert werden, unter Berücksichtigung der Unterschiede zwischen TLEBDDs und ETLEBDDs, auch für ETLEBDDs gültig.

ETLEBDDs dienen zur Repräsentation von isomorphen Transitionen mehrerer Komponenten und sie verwenden statt einer Zuordnungsliste ein Array von Zuordnungslisten. Da in einem ETLEBDD nur isomorphe Transitionen mehrerer Komponenten gespeichert werden können, müssen alle anderen in einem System vorhandenen Transitionen weiterhin mit anderen Datenstrukturen repräsentiert werden. Um die Vorteile des nicht expliziten Repräsentierens von Identitätstransformationen und der Umbenennung der Eingabevariablen auf eine reduzierte Variablenmenge bei Verifikationsexperimenten ausnutzen zu können, wurden diese Transitionen in der vorliegenden Arbeit in TLEBDDs abgespeichert. Damit wurden in solchen Fällen partitionierte Transitionsrelationen aus ETLEBDDs und TLEBDDs benutzt. Die Identitätsreduktion und die Umbenennung der Eingabevariablen ermöglichen bei Verwendung von TLEBDDs im Vergleich mit gewöhnlichen BDDs das ver-

stärkte Ausnutzen des nur einmaligen Abspeicherns von zwischen verschiedenen Entscheidungsdiagrammen vorhandenen isomorphen Teilgraphen durch BDD-Programmpakete. Bei Systemen aus replizierten Komponenten besitzen TLEBDDs zur Repräsentation der Transitionen unterschiedlicher Komponenten zum Beispiel oft sehr große isomorphe Teilgraphen. Ein ETLEBDD zur Speicherung von Transitionen mehrerer Komponenten und dessen BDD werden hingegen von vorneherein nur einmal angelegt und abgespeichert. Der BDD des ETLEBDDs wird dann zur Repräsentation der in einem ETLEBDD gespeicherten Menge von isomorphen Transitionen für alle Komponenten, für die Transitionen im ETLEBDD gespeichert sind, benutzt. Vor dem Bau eines ETLEBDDs muss eine in einem ETLEBDD zu speichernde Menge von zwischen mehreren Komponenten vorhandenen isomorphen Transitionen manuell oder automatisiert ermittelt werden (siehe dazu Abschnitt 9.2). Zusätzlich kann der BDD eines ETLEBDDs isomorphe Teilgraphen zu anderen ETLEBDDs oder TLEBDDs besitzen, die dann von BDD-Programmpaketen auch nur einmal abgespeichert werden müssen. Während in einem ETLEBDD Mengen von Transitionen verschiedener Komponenten gespeichert werden, die mit dem gleichen BDD des ETLEBDDs repräsentiert werden können, erlaubt die Verwendung von TLEBDDs das Ausnutzen von vorhandenen isomorphen Teilgraphen bei unterschiedlichen BDDs von TLEBDDs. Solche Unterschiede treten in Systemen aus replizierten Komponenten zum Beispiel bei vorhandenen id-sensitiven Variablen (siehe Abschnitt 2.4.1.1) auf, die bei Transitionen für unterschiedliche Komponenten oft auf verschiedene Werte gesetzt werden müssen. Das Ausnutzen von isomorphen Teilgraphen erfolgt dann nur für die zwischen den BDDs der TLEBDDs isomorphen Teilgraphen, was wie die in der vorliegenden Arbeit präsentierten experimentellen Ergebnisse zeigen meistens immer noch zu sehr großen Reduktionen des Speicherbedarfs führt.

Der in Abschnitt 5.3 vorgestellte Algorithmus zur simultanen Berechnung des relationalen Produkts mit einem ETLEBDD für mehrere Komponenten führt, wie der in Abschnitt 4.3 für TLEBDDs präsentierte Algorithmus, eine kombinierte Berechnung des relationalen Produkts durch. Dabei werden die im relationalen Produkt enthaltenen Operationen (UND-Verknüpfung, existentielle Quantifizierung bezüglich der Variablen für aktuelle Zustände und Umbenennung von Variablen für Nachfolgerzustände in die zu diesen korrespondierenden Variablen für aktuelle Zustände) gemeinsam berechnet. Zusätzlich wird die Zuordnung der reduzierten Variablen von Knoten des BDDs des ETLEBDDs auf die diesen entsprechenden und beim BDD für die Zustandsmenge verwendeten tatsächlichen Variablen direkt während der Berechnung des relationalen Produkts durchgeführt. Diese Zuordnung erfolgt auch im in Abschnitt 4.3 für TLEBDDs präsentierten Algorithmus direkt während der Berechnung des relationalen Produkts. Daher gelten die in Abschnitt 4.6 für die Vorgehensweise beim Algorithmus für TLEBDDs dafür durchgeführten Abgrenzungen zu verwandten Arbeiten ebenfalls für den in diesem Abschnitt für ETLEBDDs vorgestellten Algorithmus. Im Vergleich mit dem Algorithmus für TLEBDDs wird beim Algorithmus für ETLEBDDs keine Berechnung des relationalen Produkts für eine Komponente durchgeführt, sondern mit diesem Algorithmus kann das relationale Produkt für alle Komponenten, für die Transitionen in einem ETLEBDD gespeichert sind, auf einmal berechnet werden. Außerdem erlaubt der Algorithmus für ETLEBDDs zusätzlich auf

mit dem Wurzelknoten des BDDs des ETLEBDDs und dem Wurzelknoten des beteiligten BDDs zur Repräsentation einer Zustandsmenge beginnenden Berechnungspfaden das Ausnutzen von bei Berechnungen für verschiedene Komponenten auf den Berechnungspfaden in Rekursionsschritten gleich auszuführenden Teilberechnungen, in dem diese nur einmal ausgeführt werden.

Ein in Abschnitt 2.5.6 vorgestelltes Verfahren, bei dem versucht wird die Anzahl der Nachfolgerberechnungen zu reduzieren ist das des *iterative squaring*. ETLEBDDs können bei manchen Systemen die Anzahl der Nachfolgerberechnungen im Vergleich zur Benutzung von komponentenweise partitionierten Transitionsrelationen reduzieren, da mit ihnen Transitionen mehrerer Komponenten repräsentiert werden können. Nachfolgerberechnungen können dann mit dem in Abschnitt 5.3 für die Berechnung des relationalen Produkts mit einem ETLEBDD präsentierten Algorithmus mit allen in einem ETLEBDD gespeicherten Transitionen auf einmal durchgeführt werden. Beim *iterative squaring* wird der transitive Abschluss der Transitionsrelation berechnet. Mit diesem Ansatz können mit einer Nachfolgerberechnung alle in dieser Relation gespeicherten und erreichbaren Zielzustände auf einmal erreicht werden. Dadurch kann beim *iterative squaring* die Anzahl an Nachfolgerberechnungen über einzelne Explorationsrunden hinweg reduziert werden. Allerdings muss die entsprechende für Nachfolgerberechnungen zu verwendende Repräsentation von aktuellen Zuständen und aus diesen durch Ausführen und Hintereinanderausführen von Transitionen erreichbaren Zielzuständen vorher berechnet werden, was aufwendig sein kann. Bei Verwendung von ETLEBDDs kann es in Abhängigkeit vom zu verifizierenden System sein, dass die innerhalb einer Explorationsrunde auszuführende Anzahl an Nachfolgerberechnungen reduziert wird. Wie die Ergebnisse der durchgeführten Verifikationsexperimente zeigen, kann auch die Explorationsstrategie *mehrfache simultane Exploration* durch die direkte Hintereinanderausführung von Nachfolgerberechnungen mit dem ETLEBDD für mehrere Komponenten zu einer Reduktion der insgesamt auszuführenden Anzahl an Nachfolgerberechnungen führen. Außerdem ist vor dem Beginn des Ausführens von Nachfolgerberechnungen mit dem ETLEBDD keine Anpassung eines konstruierten ETLEBDDs notwendig.

Zur Benutzung von ETLEBDDs für die Erreichbarkeitsanalyse wurden in Abschnitt 5.8 verschiedene Explorationsstrategien eingeführt. Neben einer Strategie bei der ausschließlich BDDs oder TLEBDDs verwendet werden, werden dort zwei Ansätze zur Erreichbarkeitsanalyse mit einer partitionierten Transitionsrelation aus ETLEBDDs und TLEBDDs präsentiert. Beim ersten in Algorithmus 21 präsentierten Ansatz *einfache simultane Exploration* wird in einer Explorationsrunde eine Bildberechnung mit einem ETLEBDD und eine Bildberechnung mit jedem vorhandenen TLEBDD durchgeführt. Dieser Ansatz entspricht einer Breitensuche zur Exploration aller erreichbaren Zustände. Im zweiten dort vorgestellten und in Algorithmus 22 angegebenen Verfahren *mehrfache simultane Exploration* werden in einer Explorationsrunde solange Bildberechnungen mit einem ETLEBDD ausgeführt, wie damit bisher noch nicht explorierte Zustände gefunden werden können. Anschließend werden dort Bildberechnungen mit TLEBDDs durchgeführt. Durch die mehrfache Hintereinanderausführung von Bildberechnungen mit dem ETLEBDD, bevor Bildberechnungen mit den TLEBDDs durchgeführt werden, liegt hier

eine Kombination von Breiten- und Tiefensuche zur Exploration des Zustandsraums vor. Arbeiten zur Optimierung des Vorgehens bei der Zustandsraumdurchmusterung wurden in Abschnitt 2.5.7 präsentiert. Dort wurden unter anderem mehrere Verfahren, bei denen wie bei der Strategie der *mehrfachen simultanen Exploration* für ETLEBDDs eine Kombination von Breiten- und Tiefensuche durchgeführt wird, vorgestellt. Solche Verfahren wurden entwickelt, um BDDs für Zwischenergebnisse von Nachfolgerberechnungen und resultierende Zustandsmengen klein zu halten. Dazu wurden Ansätze entworfen, bei denen Teile der Transitionsrelation mit denen Nachfolgerberechnungen durchgeführt werden oder weiter zu explorierende Zustandsmengen während der Erreichbarkeitsanalyse dynamisch angepasst werden (z.B. die in Abschnitt 2.5.7 beschriebenen Verfahren aus [25, 41, 109, 165, 166, 167]). Im Gegensatz zu diesen werden bei den in Abschnitt 5.8 für ETLEBDDs vorgestellten Explorationsstrategien feste Partitionen der Transitionsrelation verwendet und es wurden auch keine Ansätze zur Variation der Menge der weiter zu explorierenden Zustände benutzt. Dafür wird bei beiden Verfahren für Nachfolgerberechnungen der für ETLEBDDs vorgestellte Algorithmus, der es ermöglicht zwischen Komponenten gleiche Teilberechnungen nur einmal auszuführen, verwendet. Bei der Explorationsstrategie *mehrfache simultane Exploration* wird dies und das Potential des Algorithmus, große Zustandsmengen für mehrere Komponenten auf einmal explorieren zu können, ausgenutzt. Dies geschieht, in dem solange Bildberechnungen mit dem ETLEBDD ausgeführt werden, bis dadurch keine noch nicht gefundenen Zustände mehr gefunden werden können.

Verfahren bei denen ebenfalls Bildberechnungen ausgeführt werden, bis Fixpunkte an Zuständen erreicht werden, sind die in Abschnitt 2.5.7 beschriebenen Ansätze aus [39, 46, 47, 48, 49, 96, 133, 151, 194]. Gründe dafür sind dort zum Beispiel die Benutzung der Vorgehensweise für ein Verfahren zur Reduktion durch Halbordnung oder zum Erreichen möglichst kompakter Repräsentationen von explorierten Zustandsmengen. Bei der für ETLEBDDs vorgestellten Explorationsstrategie der *mehrfachen simultanen Exploration* wurde die erschöpfende Bildberechnung mit dem ETLEBDD verwendet, um möglichst viele Zustände unter Ausnutzung von für mehrere Komponenten gleich auszuführenden Teilberechnungen schnell explorieren zu können. Dies erlaubt es, alle mit Bildberechnungen mit dem ETLEBDD gefundenen Zustände anschließend direkt mit TLEBDDs weiter explorieren zu können. Die direkte Weiterexploration neu gefundener Zustände in partitionierten Transitionsrelationen wird *Chaining* genannt und wurde auch in den in Abschnitt 2.5.7 beschriebenen Verfahren aus [44, 169, 179] benutzt.

6 Neues Verfahren zur Kombination von Teilresultaten

In diesem Kapitel wird ein Algorithmus zur Berechnung von Nachfolgerzuständen vorgestellt, bei dem ein neuer Ansatz zur Berücksichtigung bereits fertig berechneter Teilresultate verwendet wird. Dieser wird hier für BDDs präsentiert. Eine Version für die Benutzung von TLEBDDs zur Repräsentation von Transitionen ist im Anhang (Kapitel 10) zu finden. Außerdem werden in diesem Kapitel verschiedene Möglichkeiten zur Anwendung des neuen Algorithmus bei der Erreichbarkeitsanalyse vorgestellt.

6.1 Algorithmus mit neuer Kombination von Teilresultaten

In diesem Abschnitt wird für verschachtelte Variablenordnungen ein Algorithmus zur kombinierten Berechnung des relationalen Produkts zwischen BDDs vorgestellt. Neben der Berechnung des relationalen Produkts kann dem Algorithmus ein bereits fertig berechnetes Teilresultat übergeben werden, das er mit dem Ergebnis des relationalen Produkts vereinigt. Im ersten Aufruf des Algorithmus kann diesem als Teilresultat ein BDD mit einer bereits ermittelten Zustandsmenge übergeben werden. Diese Zustandsmenge wird vom Algorithmus dann mit dem für das relationale Produkt berechneten Resultat vereinigt. Der Algorithmus benutzt ein neues Verfahren zur Kombination bereits fertig berechneter Teilresultate mit dem Ergebnis noch ausstehender Berechnungen. Die Kombination erfolgt durch simultanes Durchlaufen von BDDs mit Teilresultaten während der rekursiven Berechnung von in einem Rekursionsschritt noch ausstehenden Ergebnissen. Bei Auftreten eines Terminalfalls in einem Rekursionsschritt wird das aktuell vorhandene Teilresultat mit dem Resultat des Rekursionsschritts vereinigt. Dadurch können die gewöhnlich verwendete explizite Vereinigung von BDDs mit Teilergebnissen durch deren Veroderung (siehe z.B. Algorithmus zur kombinierten Berechnung des relationalen Produkts aus Abschnitt 3.1) und die dazu nötigen BDD-Traversierungen vermieden werden. Das relationale Produkt für die Nachfolgerberechnung unter zusätzlicher Berücksichtigung eines BDDs *Partial($\vec{x}$)* mit einem bereits fertig berechneten Teilresultat kann wie folgt beschrieben werden:

$$\exists \vec{x}[R(\vec{x},\vec{x}') \wedge S(\vec{x})]\{\vec{x}' \leftarrow \vec{x}\} \vee Partial(\vec{x}) \tag{6.1}$$

Dabei ist $R(\vec{x},\vec{x}')$ ein BDD mit einer Menge von Transitionen, $S(\vec{x})$ ein BDD zur Repräsentation einer Zustandsmenge und wie bereits erwähnt *Partial($\vec{x}$)* ein BDD mit einem bereits fertig berechneten Teilresultat. Der Algorithmus *RelProductPartial()* zur kombinierten Berechnung des relationalen Produkts unter Berücksichtigung eines bereits fertig berechneten Teilresultats ist in Algorithmus 23 dargestellt. Das relationale Produkt wird

von diesem mit dem bei Algorithmen für BDDs üblicherweise verwendeten rekursiven Berechnungsprinzip berechnet, bei dem in Rekursionsschritten eine Aufspaltung in Teilprobleme für die Belegung der Top-Variablen mit den beiden Booleschen Werten erfolgt. Die Aufspaltung in Teilprobleme und das Durchführen weiterer rekursiver Aufrufe werden durchgeführt, wenn nach vorheriger Prüfung auf Terminalfälle kein Terminalfall erkannt wurde und das zu berechnende Ergebnis anschließend auch nicht in der Ergebnistabelle gefunden wurde. Der BDD *Partial($\vec{x}$)* mit einem bereits fertig berechneten Teilresultat kann beim ersten Aufruf des Algorithmus bei der rekursiven Berechnung des relationalen Produkts eine bereits ermittelte Zustandsmenge beinhalten. Bei späteren rekursiven Aufrufen des Algorithmus enthält der BDD *Partial($\vec{x}$)* während der rekursiven Berechnung ermittelte Teilresultate.

Algorithmus 23: Kombinierte Berechnung des relationalen Produkts zwischen BDDs, bei der die Vereinigung von Teilresultaten mit dem Ergebnis noch ausstehender Berechnungen durch simultanes Durchlaufen von BDDs mit Teilresultaten durchgeführt wird.

```
1  RelProductPartial(ROBDD R, ROBDD S, ROBDD Partial)
2  if IsTerminalCase(R, S) then
3      return TerminalCase(R, S, Partial);
4  if ComputedTableHasEntry(RelProdPartial, R, S, Partial) then
5      return ComputedTable(RelProdPartial, R, S, Partial);
6  targetVarPartial = TargetVariable(Partial);
7  v = TopVariable(R, S, targetVarPartial);
8  if IsCurrentStateVariable(v) then
9      R_1 = RelProductPartial(R|_{v=1}, S|_{v=1}, Partial);
10     R_0 = RelProductPartial(R|_{v=0}, S|_{v=0}, R_1);
11     Result = R_0;
12 else
13     w = GetCurrentVariable(v);
14     R_1 = RelProductPartial(R|_{v=1}, S, Partial|_{w=1});
15     R_0 = RelProductPartial(R|_{v=0}, S, Partial|_{w=0});
16     if R_0 == R_1 then
17         return R_1;
18     Result = FindOrAddUniqueTable(w, R_0, R_1)
19 InsertComputedTable(RelProdPartial, R, S, Partial, Result);
20 return Result;
```

Im Algorithmus *RelProductPartial()* wird der BDD *Partial($\vec{x}$)* mit einem Teilresultat während der rekursiven Berechnung des relationalen Produkts bei Rekursionsschritten mit Variablen für aktuelle Zustände und mit Variablen für Nachfolgerzustände als Top-Variablen auf unterschiedliche Arten simultan mit durchlaufen. Die Korrektheit der dabei

verwendeten Aufspaltungen in Teilprobleme und die der Kombination von dafür resultierenden Teilergebnissen zum Ergebnis eines Rekursionsschritts wird in Satz 6.1 in Abschnitt 6.2 gezeigt. Die dort bewiesenen und nachfolgend angegebenen beiden Gleichungen sind die Basis dafür, wie im Algorithmus die Aufspaltung der in einem Rekursionsschritt des Algorithmus durchzuführenden Berechnung in Teilprobleme bei einer Variable für aktuelle Zustände und bei einer Variable für Nachfolgerzustände als Top-Variable erfolgt:

- υ Variable für aktuelle Zustände ($\upsilon \in X$):

$$\begin{aligned}\exists\vec{x}[R(\vec{x},\vec{x}') \wedge S(\vec{x})]\{\vec{x}' \leftarrow \vec{x}\} \vee Partial(\vec{x}) = \\ (\exists\vec{x}((R(\vec{x},\vec{x}')|_{\upsilon=1} \wedge S(\vec{x})|_{\upsilon=1}) \vee Partial(\vec{x})\{\vec{x} \leftarrow \vec{x}'\}))\{\vec{x}' \leftarrow \vec{x}\} \vee \\ (\exists\vec{x}(R(\vec{x},\vec{x}')|_{\upsilon=0} \wedge S(\vec{x})|_{\upsilon=0}))\{\vec{x}' \leftarrow \vec{x}\}\end{aligned}$$

- υ Variable für Nachfolgerzustände ($\upsilon \in X'$):

$$\begin{aligned}\exists\vec{x}[R(\vec{x},\vec{x}') \wedge S(\vec{x})]\{\vec{x}' \leftarrow \vec{x}\} \vee Partial(\vec{x}) = \upsilon\{\vec{x}' \leftarrow \vec{x}\} \cdot (\exists\vec{x}((R(\vec{x},\vec{x}')|_{\upsilon=1} \wedge \\ S(\vec{x})|_{\upsilon=1}) \vee Partial(\vec{x})|_{(\upsilon\{\vec{x}' \leftarrow \vec{x}\})=1}\{\vec{x} \leftarrow \vec{x}'\}))\{\vec{x}' \leftarrow \vec{x}\} \vee \overline{\upsilon}\{\vec{x}' \leftarrow \vec{x}\} \cdot \\ (\exists\vec{x}((R(\vec{x},\vec{x}')|_{\upsilon=0} \wedge S(\vec{x})|_{\upsilon=0}) \vee Partial(\vec{x})|_{(\upsilon\{\vec{x}' \leftarrow \vec{x}\})=0}\{\vec{x} \leftarrow \vec{x}'\}))\{\vec{x}' \leftarrow \vec{x}\}\end{aligned}$$

An den Gleichungen und im Algorithmus *RelProductPartial()* ist zu sehen, dass der BDD *Partial($\vec{x}$)* in einem Rekursionsschritt mit einer Variablen für aktuelle Zustände als Top-Variable unverändert, als Übergabeparameter beim rekursiven Aufruf für das der Belegung der Top-Variable mit dem Booleschen Wert 1 entsprechende Teilproblem verwendet wird. Das für das Teilproblem ermittelte Ergebnis ist wieder ein fertig berechnetes Teilresultat. Dieses wird im Algorithmus *RelProductPartial()* daher hier gleich wieder als Übergabeparameter für ein fertig berechnetes Teilresultat beim rekursiven Aufruf des Algorithmus für das Teilproblem für eine Belegung der Top-Variable mit dem Booleschen Wert 0 benutzt. Die Korrektheit dieses Vorgehens kann durch entsprechende Umformung der oben für eine Variable für aktuelle Zustände angegebenen Aufspaltung in Teilprobleme gezeigt werden. Bei einer Top-Variable für Nachfolgerzustände wird der BDD *Partial($\vec{x}$)*, der nur Knoten für Variablen für aktuelle Zustände besitzt da er ein bereits fertig berechnetes Teilresultat repräsentiert, simultan mit durchlaufen. Es wird dabei bei einem rekursiven Aufruf des Algorithmus für ein durch Belegung der Top-Variable mit einem Booleschen Wert gewonnenes Teilproblem, ein durch Belegung der der Top-Variable entsprechenden Variablen für aktuelle Zustände mit dem gleichen Booleschen Wert ermittelter Kofaktor des BDDs *Partial($\vec{x}$)* verwendet. Die hier durch die rekursiven Aufrufe für Teilprobleme für die beiden Belegungen der Top-Variable ermittelten Teilergebnisse werden anschließend zum Rückgabewert des Rekursionsschritts kombiniert.

Im Folgenden wird der in Algorithmus 23 angegebene Algorithmus *RelProductPartial()* genauer beschrieben. Der Algorithmus führt eine kombinierte Berechnung des relationalen Produkts durch und berechnet die im relationalen Produkt enthaltenen Operationen auf einmal. Als Übergabeparameter besitzt er einen BDD R mit einer Menge von Transitionen, einen BDD S zur Repräsentation der zu explorierenden Zustandsmenge und einen

BDD *Partial* mit einem bereits fertig berechneten Teilresultat. Ein in einem Rekursionsschritt vorhandener BDD mit einem Teilresultat wird dabei vom Algorithmus mit dem in einem Rekursionsschritt für das relationale Produkt zu berechnenden Ergebnis vereinigt. Zuerst wird im Algorithmus, wie im gewöhnlichen Algorithmus zur kombinierten Berechnung des relationalen Produkts mit BDDs (siehe Abschnitt 3.1) geprüft, ob ein Terminalfall der rekursiven Berechnung vorliegt. Bei einem aufgetretenen Terminalfall wird der entsprechende Rückgabewert ermittelt und zurückgegeben (Zeilen 2 und 3 in Algorithmus 23). Die im Algorithmus verwendeten Terminalfälle und deren Rückgabewerte sind in Algorithmus 24, in dem die Funktion *TerminalCase()* dargestellt ist, aufgeführt. Im Vergleich mit der entsprechenden Funktion bei der gewöhnlichen kombinierten Berechnung des relationalen Produkts bekommt sie den BDD *Partial*, mit dem das aktuell zu berücksichtigende Teilresultat repräsentiert wird, als zusätzlichen Parameter übergeben. Dies ist notwendig, da das durch diesen BDD repräsentierte Teilresultat mit dem Ergebnis eines Rekursionsschritts zu vereinigen ist.

Algorithmus 24: Funktion *TerminalCase()*, die bei einem in Algorithmus 23 aufgetretenen Terminalfall den BDD mit dem Ergebnis des Terminalfalls berechnet und zurückgibt.

```
1 TerminalCase(ROBDD R, ROBDD S, ROBDD Partial)
2 if IsTerminalVertexZero(R) || IsTerminalVertexZero(S) then
3 |   return Partial;
4 if IsTerminalVertexOne(R) || R==S then
5 |   return BDDTerminalVertexOne;
```

Die ersten beiden Terminalfälle treten auf, wenn der Wurzelknoten des BDDs R oder der Wurzelknoten des BDDs S ein Terminalknoten für den Booleschen Wert 0 ist. Beim in Abschnitt 3.1 vorgestellten Algorithmus zur kombinierten Berechnung des relationalen Produkts zwischen BDDs ergibt sich als Rückgabewert für diese Terminalfälle der BDD für den Terminalknoten mit dem Wert 0 (siehe Algorithmus 10 in Abschnitt 3.1). Hier muss noch der BDD *Partial* mit einem bereits fertig berechneten Teilresultat berücksichtigt und mit dem Ergebnis eines Rekursionsschritts vereinigt werden. Aus dieser Vereinigung ergibt sich der BDD *Partial* in diesen beiden Fällen als Rückgabewert des in diesem Abschnitt präsentierten Algorithmus (siehe Zeilen 2 und 3 in Algorithmus 24). Als weitere Terminalfälle werden der Fall, in dem der BDD R ein Terminalknoten für den Wert 1 ist und der Fall, in dem dieser BDD und der BDD S die gleichen Mengen repräsentieren ($R == S$), verwendet. Diese wurden ebenfalls bereits beim in Abschnitt 3.1 zur Berechnung des relationalen Produkts vorgestellten Algorithmus benutzt. Als Rückgabewert wurde für diese dort der Terminalknoten für den Booleschen Wert 1 verwendet. Dieser Rückgabewert wird bei diesen beiden Terminalfällen auch beim Algorithmus *RelProductPartial()* zurückgegeben (siehe Zeilen 4 und 5 in Algorithmus 24). Der Inhalt des BDDs *Partial* ist bei Rückgabe des Terminalknotens für den Wert 1, bei dem das Teilergebnis des relationalen Produkts für alle Variablenbelegungen der in der Variablenordnung auf dem

aktuellen Berechnungspfad noch nicht betrachteten tatsächlichen Variablen auf den Wert 1 auswertet, schon enthalten. Deshalb ist bei diesen beiden Terminalfällen keine Änderung des Rückgabewerts notwendig. Eine etwas ausführlichere Erläuterung der Korrektheit der im Algorithmus *RelProductPartial()* verwendeten Terminalfälle ist in Abschnitt 6.3 zu finden.

Ist in einem Rekursionsschritt kein Terminalfall vorhanden wird überprüft, ob das Ergebnis der auszuführenden Berechnung bereits vorher ermittelt wurde und noch in der Ergebnistabelle gespeichert ist (Zeilen 4 und 5 in Algorithmus 23). Als Übergabeparameter für den Zugriff auf die Ergebnistabelle werden hier die schon bei der gewöhnlichen Berechnung des relationalen Produkts verwendeten Parameter benutzt. Zusätzlich wird der Wurzelknoten des BDDs *Partial* mit dem Teilresultat als Übergabeparameter verwendet. Dies ist notwendig, um die Eindeutigkeit von Zugriffen auf die Ergebnistabelle sicherstellen, da sich für Rekursionsschritte mit den gleichen BDDs R und S aber unterschiedlichem BDD *Partial* verschiedene Rückgabewerte des Algorithmus ergeben können. Daher wird der BDD *Partial* als zusätzlicher Übergabeparameter verwendet und in den Einträgen der Ergebnistabelle mit abgespeichert. Für einen erfolgreichen Zugriff auf die Ergebnistabelle muss der Wert dieses Übergabeparameters mit dem in einem Eintrag der Ergebnistabelle für den BDD *Partial* gespeicherten Wert übereinstimmen. Bei Übereinstimmung der bei Zugriffen auf die Ergebnistabelle hier verwendeten Übergabeparameter ist das Ergebnis des relationalen Produkts unter Berücksichtigung eines Teilresultats eindeutig bestimmt, wodurch die Eindeutigkeit der aus der Ergebnistabelle erhaltenen Rückgabewerte sichergestellt ist.

Konnte das zu berechnende Ergebnis nicht in der Ergebnistabelle gefunden werden, wird die Top-Variable des aktuellen Rekursionsschritts bezüglich der verwendeten Variablenordnung ermittelt. Als Übergabeparameter für ein zu berücksichtigendes Teilresultat werden dem Algorithmus nur von Variablen für aktuelle Zustände abhängige BDDs ($Partial(\vec{x})$) übergeben, da in BDDs für Ergebnisse und Teilergebnisse der Berechnung des relationalen Produkts nur Variablen für aktuelle Zustände wesentliche Variablen sind. Der Grund dafür ist die im relationalen Produkt enthaltene existentielle Quantifizierung bezüglich der Variablen für aktuelle Zustände und die ebenfalls durchzuführende Umbenennung von Knoten für Variablen für Nachfolgerzustände in Knoten für entsprechende Variablen für aktuelle Zustände. Daher wurde für Teilergebnisse die im relationalen Produkt enthaltene existentielle Quantifizierung bezüglich der Variablen für aktuelle Zustände und die Umbenennung von Knoten für Variablen für Nachfolgerzustände, in Knoten für entsprechende Variablen für aktuelle Zustände, bereits durchgeführt. Aufgrund der Umbenennungsoperation geben bei Nachfolgerberechnungen die im BDD R zur Repräsentation von Transitionen für Variablen für Nachfolgerzustände möglichen Werte, die möglichen Werte für Variablen für aktuelle Zustände im Ergebnis des relationalen Produkts an. Für einen BDD mit einem fertig berechneten Teilresultat beeinflussen hingegen mögliche Werte für Variablen für aktuelle Zustände das Ergebnis des relationalen Produkts unter Berücksichtigung eines fertig berechneten Teilresultats. Daher müssen Knoten für Variablen für aktuelle Zustände im BDD mit einem Teilresultat im hier vorgestellten Algorithmus beim simultanen Durchlaufen des BDDs mit dem Teilresultat gemeinsam mit

Knoten für Variablen für Nachfolgerzustände im BDD zur Repräsentation von Transitionen betrachtet werden. Deshalb werden die Knoten des BDD *Partial* für Variablen für aktuelle Zustände im Algorithmus als Knoten für die zu diesen korrespondierenden Variablen für Nachfolgerzustände berücksichtigt. In Zeile 6 des Algorithmus wird die Variable für Nachfolgerzustände zur Variablen für aktuelle Zustände des Wurzelknotens des BDDs *Partial* ermittelt. Diese wird nachfolgend bei der Ermittlung der Top-Variablen eines Rekursionsschritts in Zeile 7 des Algorithmus mit berücksichtigt. Zu berechnende Kofaktoren des BDDs *Partial* müssen aber trotzdem bezüglich der zu einer Top-Variablen für Nachfolgerzustände korrespondierenden Variablen für aktuelle Zustände gebildet werden, da im BDD *Partial* nur Knoten für Variablen für aktuelle Zustände enthalten sind. Daher wird die Variable für aktuelle Zustände zu einer Top-Variablen v für Nachfolgerzustände vor der Ermittlung von Kofaktoren des BDDs *Partial* in einem Rekursionsschritt durch die Funktion *GetCurrentVariable()* ermittelt (siehe Zeile 13).

Das Ergebnis der in einem Rekursionsschritt auszuführenden Berechnung wird nun durch Aufspaltung der Berechnung in einfachere Teilprobleme und deren Lösung in weiteren rekursiven Aufrufen des Algorithmus bestimmt. Dabei bestehen die einfacheren Teilprobleme aus Kofaktoren der BDDs R, S und *Partial*. Die Aufspaltung der in einem Rekursionsschritt durchzuführenden Berechnung in Teilprobleme erfolgt für Rekursionsschritte mit Variablen für aktuelle Zustände und für solche mit Variablen für Nachfolgerzustände als Top-Variablen wie oben beschrieben auf unterschiedliche Weise. Für die beiden im Algorithmus dabei verwendeten Aufspaltungen und für die Kombination von ermittelten Teilergebnissen zum Rückgabewert eines Rekursionsschritts wird die Korrektheit in Abschnitt 6.2 gezeigt. Bei einer Top-Variablen für aktuelle Zustände wird der BDD *Partial* beim rekursiven Aufruf des Algorithmus mit den Kofaktoren bezüglich des Werts 1 der Top-Variablen als Übergabeparameter verwendet (siehe Zeile 9). Falls im BDD *Partial* für diese Top-Variable ein Knoten vorhanden ist muss dieser Knoten, wie oben beschrieben, als Knoten für die zur Variablen für aktuelle Zustände korrespondierende Variable für Nachfolgerzustände berücksichtigt werden. Bei verschachtelten Variablenordnungen, für die der hier beschriebene Algorithmus entwickelt wurde, sind für ein Variablenpaar aus einer Variable für aktuelle Zustände und der korrespondierenden Variablen für Nachfolgerzustände zwei Anordnungen in einer Variablenordnung möglich. Es kann die Variable für aktuelle Zustände vor der korrespondierenden Variablen für Nachfolgerzustände in der Variablenordnung vorkommen oder die beiden Variablen können in umgekehrter Reihenfolge vorkommen. In einem Rekursionsschritt mit einer Variablen für aktuelle Zustände als Top-Variable kann der Wurzelknoten des BDDs *Partial* nur ein Knoten für diese Variable für aktuelle Zustände sein, wenn die zur Top-Variablen korrespondierende Variable für Nachfolgerzustände nach der Variablen für aktuelle Zustände in der Variablenordnung vorkommt. In der zweiten Anordnung der Variablen kommt die Variable für Nachfolgerzustände vor der korrespondierenden Top-Variable für aktuelle Zustände in der Variablenordnung vor. Hier würde ein für die Variable für aktuelle Zustände im BDD mit dem Teilresultat vorhandener Wurzelknoten bereits beim vorher für die Variable für Nachfolgerzustände auszuführenden Rekursionsschritt berücksichtigt werden. Damit gleicht die Variable des Wurzelknotens des BDDs *Partial* der Top-Variablen für

aktuelle Zustände oder sie kommt später in der Variablenordnung vor. In beiden Fällen müssen im BDD *Partial* vorhandene Knoten in nachfolgenden Rekursionsschritten und zusammen mit Knoten für später in der Variablenordnung vorkommende Variablen evaluiert werden. Das gilt auch für das in Zeile 9 ermittelte Teilresultat R_1. Dieses wird als Übergabeparameter für ein bereits fertig berechnetes Teilresultat beim rekursiven Aufruf des Algorithmus *RelProductPartial()* für das Teilproblem aus den Kofaktoren bezüglich des Werts 0 der Top-Variable verwendet (siehe Zeile 10). Im mit diesem rekursiven Aufruf ermittelten Teilergebnis R_0 ist das Ergebnis der in diesem Rekursionsschritt für das relationale Produkt durchzuführenden Berechnung enthalten und auch der Inhalt des BDDs *Partial* wurde berücksichtigt. Wie nachfolgend in Abschnitt 6.2 gezeigt wird, ist dies direkt der Rückgabewert des Algorithmus für einen solchen Rekursionsschritt.

Der BDD *Partial* enthält nur Knoten für Variablen für aktuelle Zustände, die wie oben beschrieben im Algorithmus beim simultanen Durchlaufen als Knoten für die zu diesen korrespondierenden Variablen für Nachfolgerzustände berücksichtigt werden. Daher werden bei den rekursiven Aufrufen des Algorithmus in Rekursionsschritten mit einer Variablen für Nachfolgerzustände als Top-Variable Kofaktoren des BDDs *Partial*, bezüglich der der Top-Variable entsprechenden Variablen für aktuelle Zustände (w, siehe Zeile 13), als Übergabeparameter für bereits fertig berechnete Teilresultate verwendet (siehe Zeilen 14 und 15). In Rekursionsschritten mit einer Variablen für Nachfolgerzustände als Top-Variable wird auch die im relationalen Produkt enthaltene Umbenennung von Variablen für Nachfolgerzustände, in deren korrespondierende Variablen für aktuelle Zustände, durchgeführt. Dazu wird eines der beiden rekursiv ermittelten Teilresultate R_1 oder R_0 zurückzugeben, wenn die beiden Teilresultate gleich sind (im Algorithmus wird R_1 verwendet, siehe Zeilen 16 und 17). Das Ergebnis der Berechnung eines solchen Rekursionsschritts ist dann unabhängig von der Belegung der Top-Variablen gleich, weshalb im BDD des Resultats hier kein Knoten für die Top-Variable hinzugefügt werden muss. Bei unterschiedlichen Teilergebnissen R_1 und R_0 wird ein neuer Knoten für die zur Top-Variablen korrespondierende Variable für aktuelle Zustände erzeugt. Dadurch wird die im relationalen Produkte enthaltene Umbenennung von Variablen für Nachfolgerzustände in deren korrespondierende Variablen für aktuelle Zustände durchgeführt. Die korrespondierende Variable für aktuelle Zustände w zur Top-Variablen wurde bereits in Zeile 13 ermittelt und der neue Knoten wird von der Funktion *FindOrAddUniqueTable()* (siehe Algorithmus 3 in Abschnitt 2.2.4) zurückgeliefert. Die Korrektheit der für eine Top-Variable für Nachfolgerzustände beschriebenen Kombination der Teilergebnisse zum Ergebnis eines Rekursionsschritts kann direkt aus der in Satz 6.1 im nachfolgenden Abschnitt dafür angegebenen Aufspaltung in Teilprobleme abgeleitet werden. Im relationalen Produkt ist auch die existentielle Quantifizierung bezüglich Variablen für aktuelle Zustände auszuführen, was hier bei Rekursionsschritten mit einer Variablen für aktuelle Zustände als Top-Variable erfolgt. Beim Algorithmus zur kombinierten Berechnung des relationalen Produkts aus Abschnitt 3.1 werden die in einem Rekursionsschritt ermittelten Teilergebnisse dazu explizit vereinigt. Im hier vorgestellten Algorithmus wird die zur existentiellen Quantifizierung notwendige Vereinigung durch simultanes Durchlaufen von BDDs mit Teilresultaten und die Anpassung der Terminalfälle

durchgeführt. Nach der Ermittlung des Ergebnisses eines Rekursionsschritts wird dieses in die Ergebnistabelle eingetragen (siehe Zeile 19) und zurückgegeben (siehe Zeile 20).

Im Folgenden wird der Ablauf des in diesem Abschnitt vorgestellten Algorithmus mit Hilfe des in Abschnitt 2.6 eingeführten Beispielsystems veranschaulicht. Da damit die Vorgehensweise des Algorithmus bei der rekursiven Berechnung aufgezeigt werden soll, wird der Ablauf des Algorithmus ohne Benutzung der Ergebnistabelle beschrieben. Als Übergabeparameter für den *1. Aufruf* des Algorithmus *RelProductPartial()* im Beispielablauf sind in der nachfolgend aufgeführten Abbildung 6.1 ein Ausschnitt des in Abbildung 2.13 in Abschnitt 2.6 für Komponente 2 angegebenen BDDs mit Transitionen (in Abbildung 2.13 gekennzeichnet mit *TR1*), ein Ausschnitt des dort in Abbildung 2.14 angegebenen BDDs zur Repräsentation des Anfangszustands des Systems (in Abbildung 2.14 gekennzeichnet mit *A1*) und ein BDD für ein zu berücksichtigendes Teilresultat angegeben. Als Variablenordnung wird für die BDDs die in Abschnitt 2.6 mit dem Beispielspielsystem verwendete Variablenordnung $<_{level_3}$, mit $x_g <_{level_3} x'_g <_{level_3} x_1 <_{level_3} x'_1 <_{level_3} x_2 <_{level_3} x'_2$, benutzt. Mit den dem *1. Aufruf* des Algorithmus als Übergabeparameter übergebenen BDDs ergibt sich die Variable x_2 zur Kodierung von aktuellen Zuständen als Top-Variable des Rekursionsschritts. Der Wurzelknoten des BDDs *Partial* mit dem zu berücksichtigenden Teilresultat ist ebenfalls ein Knoten für die Top-Variable x_2. Da die Variable x_2 eine Variable für aktuelle Zustände ist, wird dieser Knoten in Rekursionsschritten des Algorithmus *RelProductPartial()* aber wie oben beschrieben als Knoten für die zu dieser Variablen korrespondierende Variable für Nachfolgerzustände x'_2 berücksichtigt. Die weiteren rekursiven Aufrufe des Algorithmus *RelProductPartial()* für Teilprobleme werden anschließend, wie in den Zeilen 9 und 10 in Algorithmus 23 für eine Variable für aktuelle Zustände als Top-Variable angegeben, durchgeführt. Die Variable x_2 des Wurzelknotens des BDDs *Partial* wird im Algorithmus erst mit der zu dieser korrespondierenden Variable für Nachfolgerzustände x'_2 berücksichtigt. Daher wird der BDD *Partial* unverändert als Übergabeparameter für das zusätzlich zu berücksichtigende Teilresultat an den *2. Aufruf* des Algorithmus übergeben. Im *2. Aufruf* des Algorithmus ist direkt ein Terminalfall vorhanden, da die übergebenen BDDs R und S Terminalknoten für den Booleschen Wert 0 sind. Wie in Algorithmus 24 angegeben (Zeilen 2 und 3 in Algorithmus 24), ist der BDD mit dem übergebenen Teilresultat daher der Rückgabewert dieses Rekursionsschritts. Nach Ermittlung des Rückgabewerts des *2. Aufrufs* des Algorithmus, wird der rekursive Aufruf für das Teilproblem aus den Kofaktoren bezüglich des Werts 0 der Top-Variable x_2 des *1. Aufrufs* des Algorithmus durchgeführt. Im diesem entsprechenden *3. Aufruf* des Algorithmus ist, wie im Algorithmus *RelProductPartial()* in Zeile 10 angegeben, der Rückgabewert des *2. Aufrufs* des Algorithmus (R_1) der Übergabeparameter für ein bereits fertig berechnetes Teilresultat. Top-Variable dieses Rekursionsschritts ist die Variable x'_2 zur Kodierung von Nachfolgerzuständen. Daher werden im *3. Aufruf* des Algorithmus die in den Zeilen 14 und 15 in Algorithmus 23 angegebenen weiteren rekursiven Aufrufe ausgeführt. Für diese werden Kofaktoren des BDDs *Partial*, bezüglich der zur Top-Variable korrespondierenden Variable für aktuelle Zustände x_2, berechnet und als Übergabeparameter verwendet. In den im *3. Aufruf* des Algorithmus durchgeführten beiden rekursiven Aufrufen sind jeweils direkt Terminalfälle vorhanden. Der Wurzelknoten des BDDs R ist im *4. Aufruf* des Algo-

rithmus ein Terminalknoten für den Booleschen Wert 1, wodurch dieser Terminalknoten auch der Rückgabewert dieses Rekursionsschritts ist. Im *5. Aufruf* des Algorithmus ist der Wurzelknoten des BDDs R ein Terminalknoten für den Wert 0, wodurch der BDD *Partial* dieses Rekursionsschritts ($R_1|_{x_2=0}$) der Rückgabewert des Rekursionsschritts ist (siehe jeweils Algorithmus 24).

Der Terminalknoten für den Booleschen Wert 1 ist der Rückgabewert des *4.* und *5. Aufrufs* des Algorithmus. Daher ist dieser für beide Teilprobleme für Belegungen der Top-Variable x'_2 des *3. Aufrufs* des Algorithmus gleich. Mit der in den Zeilen 16 und 17 im Algorithmus *RelProductPartial()* dafür angegebenen Resultatskombination, bei der eines der beiden Teilresultate zurückgegeben wird, ergibt sich dieser Terminalknoten auch als Rückgabewert des *3. Aufrufs*. Dieser Terminalknoten ist damit das in Zeile 10 des Algorithmus im *1. Aufruf* des Algorithmus ermittelte Teilresultat R_0. Das Teilresultat ist dadurch das Ergebnis der Berechnung des relationalen Produkts unter Berücksichtigung eines Teilresultats im Beispiel, das in Zeile 20 im *1. Aufruf* von Algorithmus 23 zurückgegeben wird.

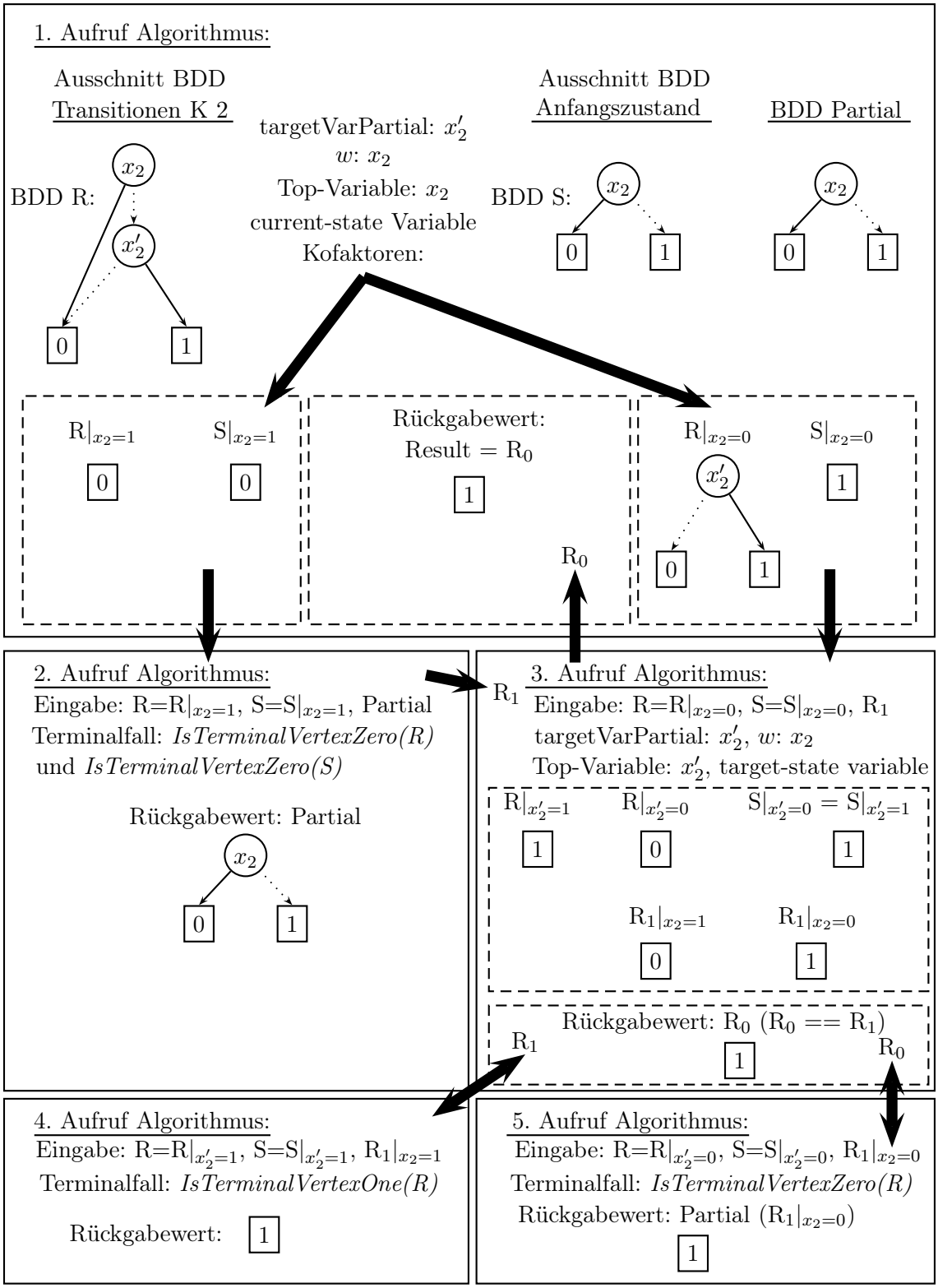

Abbildung 6.1: Beispiel zur Bildberechnung mit dem Algorithmus mit dem neuen Verfahren zur Kombination von Teilresultaten.

6.2 Korrektheit des Algorithmus

In diesem Abschnitt wird die Korrektheit der im Algorithmus aus Abschnitt 6.1 verwendeten Aufspaltungen in Teilprobleme und der dabei erfolgenden Berücksichtigung bereits fertig berechneter Teilresultate bewiesen. Da die verwendeten Terminalfälle korrekte Rückgabewerte besitzen (siehe Abschnitt 6.3) und auch bei Zugriffen auf die Ergebnistabelle nur richtige Resultate zurückgegeben werden, folgt daraus insgesamt die Korrektheit der Vorgehensweise des im vorangehenden Abschnitt vorgestellten Algorithmus *RelProduct-Partial()*. Die Grundlage zur Korrektheit der in diesem Algorithmus verwendeten Aufspaltungen in Teilprobleme wird in Satz 6.1 gezeigt.

Satz 6.1. *Gegeben sei das relationale Produkt unter Berücksichtigung eines bereits fertig berechneten Teilresultats* $\exists\vec{x}[R(\vec{x},\vec{x}')\wedge S(\vec{x})]\{\vec{x}'\leftarrow\vec{x}\}\vee Partial(\vec{x})$*, wie es in Abschnitt 6.1 eingeführt wurde. Sei außerdem* v *eine Boolesche Variable aus einer der Variablenmengen* X *und* X'*. Dann kann die Berechnung von* $\exists\vec{x}[R(\vec{x},\vec{x}')\wedge S(\vec{x})]\{\vec{x}'\leftarrow\vec{x}\}\vee Partial(\vec{x})$ *in Abhängigkeit davon, ob* v *eine Variable aus* X *oder* X' *ist, folgendermaßen aufgespalten werden:*

$$\exists\vec{x}[R(\vec{x},\vec{x}')\wedge S(\vec{x})]\{\vec{x}'\leftarrow\vec{x}\}\vee Partial(\vec{x}) =$$

$$\begin{cases} (\exists\vec{x}((R(\vec{x},\vec{x}')|_{v=1}\wedge S(\vec{x})|_{v=1})\vee Partial(\vec{x})\{\vec{x}\leftarrow\vec{x}'\}))\{\vec{x}'\leftarrow\vec{x}\}\vee \\ (\exists\vec{x}(R(\vec{x},\vec{x}')|_{v=0}\wedge S(\vec{x})|_{v=0}))\{\vec{x}'\leftarrow\vec{x}\} & v\in X \\ \\ v\{\vec{x}'\leftarrow\vec{x}\}\cdot(\exists\vec{x}((R(\vec{x},\vec{x}')|_{v=1}\wedge S(\vec{x})|_{v=1}) \\ \vee Partial(\vec{x})|_{(v\{\vec{x}'\leftarrow\vec{x}\})=1}\{\vec{x}\leftarrow\vec{x}'\}))\{\vec{x}'\leftarrow\vec{x}\}\vee\overline{v}\{\vec{x}'\leftarrow\vec{x}\}\cdot & \\ (\exists\vec{x}((R(\vec{x},\vec{x}')|_{v=0}\wedge S(\vec{x})|_{v=0})\vee Partial(\vec{x})|_{(v\{\vec{x}'\leftarrow\vec{x}\})=0}\{\vec{x}\leftarrow\vec{x}'\}))\{\vec{x}'\leftarrow\vec{x}\} & v\in X' \end{cases}$$

Beweis.

$$\exists\vec{x}[R(\vec{x},\vec{x}')\wedge S(\vec{x})]\{\vec{x}'\leftarrow\vec{x}\}\vee Partial(\vec{x})$$

$$\overset{Satz\ 3.2}{=}\begin{cases} (\exists\vec{x}(R(\vec{x},\vec{x}')|_{v=1}\wedge S(\vec{x})|_{v=1}))\{\vec{x}'\leftarrow\vec{x}\}\vee \\ (\exists\vec{x}(R(\vec{x},\vec{x}')|_{v=0}\wedge S(\vec{x})|_{v=0}))\{\vec{x}'\leftarrow\vec{x}\}\vee Partial(\vec{x}) & v\in X \\ \\ v\{\vec{x}'\leftarrow\vec{x}\}\cdot(\exists\vec{x}(R(\vec{x},\vec{x}')|_{v=1}\wedge S(\vec{x})|_{v=1}))\{\vec{x}'\leftarrow\vec{x}\}\vee & \\ \overline{v}\{\vec{x}'\leftarrow\vec{x}\}\cdot(\exists\vec{x}(R(\vec{x},\vec{x}')|_{v=0}\wedge S(\vec{x})|_{v=0}))\{\vec{x}'\leftarrow\vec{x}\}\vee Partial(\vec{x}) & v\in X' \end{cases}$$

$$= \begin{cases} (\exists\vec{x}(R(\vec{x},\vec{x}')|_{v=1} \wedge S(\vec{x})|_{v=1}))\{\vec{x}' \leftarrow \vec{x}\} \vee (Partial(\vec{x})\{\vec{x} \leftarrow \vec{x}'\})\{\vec{x}' \leftarrow \vec{x}\} & v \in X \\ \vee\ (\exists\vec{x}(R(\vec{x},\vec{x}')|_{v=0} \wedge S(\vec{x})|_{v=0}))\{\vec{x}' \leftarrow \vec{x}\} & \\ & \\ v\{\vec{x}' \leftarrow \vec{x}\} \cdot (\exists\vec{x}(R(\vec{x},\vec{x}')|_{v=1} \wedge S(\vec{x})|_{v=1}))\{\vec{x}' \leftarrow \vec{x}\} \vee & \\ \overline{v}\{\vec{x}' \leftarrow \vec{x}\} \cdot (\exists\vec{x}(R(\vec{x},\vec{x}')|_{v=0} \wedge S(\vec{x})|_{v=0}))\{\vec{x}' \leftarrow \vec{x}\} \vee v\{\vec{x}' \leftarrow \vec{x}\} \cdot & v \in X' \\ ((Partial(\vec{x})|_{(v\{\vec{x}' \leftarrow \vec{x}\})=1}\{\vec{x} \leftarrow \vec{x}'\})\{\vec{x}' \leftarrow \vec{x}\}) \vee \overline{v}\{\vec{x}' \leftarrow \vec{x}\} \cdot & \\ ((Partial(\vec{x})|_{(v\{\vec{x}' \leftarrow \vec{x}\})=0}\{\vec{x} \leftarrow \vec{x}'\})\{\vec{x}' \leftarrow \vec{x}\}) & \end{cases}$$

$$\overset{\substack{Lemma \\ 2.42}}{=} \begin{cases} (\exists\vec{x}((R(\vec{x},\vec{x}')|_{v=1} \wedge S(\vec{x})|_{v=1}) \vee Partial(\vec{x})\{\vec{x} \leftarrow \vec{x}'\}))\{\vec{x}' \leftarrow \vec{x}\} \vee & v \in X \\ (\exists\vec{x}(R(\vec{x},\vec{x}')|_{v=0} \wedge S(\vec{x})|_{v=0}))\{\vec{x}' \leftarrow \vec{x}\} & \\ & \\ v\{\vec{x}' \leftarrow \vec{x}\} \cdot (\exists\vec{x}((R(\vec{x},\vec{x}')|_{v=1} \wedge S(\vec{x})|_{v=1}) & \\ \vee\ Partial(\vec{x})|_{(v\{\vec{x}' \leftarrow \vec{x}\})=1}\{\vec{x} \leftarrow \vec{x}'\}))\{\vec{x}' \leftarrow \vec{x}\} \vee \overline{v}\{\vec{x}' \leftarrow \vec{x}\} \cdot & v \in X' \\ (\exists\vec{x}((R(\vec{x},\vec{x}')|_{v=0} \wedge S(\vec{x})|_{v=0}) \vee Partial(\vec{x})|_{(v\{\vec{x}' \leftarrow \vec{x}\})=0}\{\vec{x} \leftarrow \vec{x}'\}))\{\vec{x}' \leftarrow \vec{x}\} & \end{cases}$$

□

Da der BDD *Partial* im Algorithmus nur fertig berechnete Teilresultate des relationalen Produkts enthält, besitzt dieser nur Knoten für Variablen für aktuelle Zustände. In Satz 6.1 wurde gezeigt, wie die Berechnung des relationalen Produkts unter zusätzlicher Berücksichtigung eines Teilresultats bezüglich einer Variable für aktuelle Zustände in Teilberechnungen aufgespalten werden kann. Dabei wird der BDD *Partial* bei der Teilberechnung für eine Belegung der Variable für aktuelle Zustände mit dem Booleschen Wert 1 mit berücksichtigt. Es sind die Variablen für aktuelle Zustände, über denen der BDD *Partial* definiert ist, durch die für diese durchzuführende Umbenennungsoperation als korrespondierende Variablen für Nachfolgerzustände zu berücksichtigen ($Partial(\vec{x})\{\vec{x} \leftarrow \vec{x}'\}$). In Rekursionsschritten von Algorithmus 23 mit Variablen für aktuelle Zustände als Top-Variablen wird, wie in der dafür im Satz gezeigten Aufspaltung in Teilprobleme, verfahren. Der BDD *Partial* wird als Teilresultat beim rekursiven Aufruf des Algorithmus für das Teilproblem mit den Kofaktoren, bezüglich einer Belegung der Top-Variable mit dem Booleschen Wert 1, verwendet (siehe Zeile 9). Zusätzlich wird das Ergebnis (R_1) dieses rekursiven Aufrufs beim rekursiven Aufruf für das aus einer Belegung der Top-Variable mit dem Booleschen Wert 0 resultierende Teilproblem als bereits fertig berechnetes Teilresultat mit übergeben. Das Teilergebnis R_1 ist hier wieder ein fertig berechnetes Teilresultat, in dem nur Variablen für aktuelle Zustände wesentliche Variablen sind. Da diese im Algorithmus für deren korrespondierende Variablen für Nachfolgerzustände zu berücksichtigen sind, ist dieses Vorgehen ebenfalls korrekt. Gezeigt werden kann dies durch entsprechende Umformung der in Satz 6.1 für eine Variable für aktuelle Zustände angegebenen Aufspaltung.

In Rekursionsschritten des Algorithmus *RelProductPartial()* mit Variablen für Nachfolgerzustände als Top-Variablen wird die rekursive Aufspaltung in Teilprobleme, genau wie in Satz 6.1 für solche Variablen angegeben, durchgeführt. Knoten des BDDs *Partial* für Variablen für aktuelle Zustände werden bei der Auswahl der Top-Variable eines Re-

kursionsschritts als Knoten für deren korrespondierende Variablen für Nachfolgerzustände berücksichtigt. Ebenso werden Kofaktoren des BDDs *Partial*, bezüglich der zu einer Top-Variable für Nachfolgerzustände korrespondierenden Variablen für aktuelle Zustände, gebildet. Diese Kofaktoren werden anschließend bei den für die Belegung der Top-Variable mit den beiden Booleschen Werten ausgeführten rekursiven Aufrufen mit übergeben ($Partial|_{w=1}$ bzw. $Partial|_{w=0}$). Zur Kombination der ermittelten Teilergebnisse ist in den in Satz 6.1 präsentierten Aufspaltungen die Verundung mit der durch Umbenennung ermittelten korrespondierenden Variablen für aktuelle Zustände zur Top-Variablen durchzuführen ("$v\{\vec{x}' \leftarrow \vec{x}\}$·"bzw. "$\overline{v}\{\vec{x}' \leftarrow \vec{x}\}$·"), bevor die Teilresultate verodert werden. Dies entspricht bei gleichen Teilergebnissen (R_1 und R_0) der Rückgabe eines der Teilergebnisse. Sind die Teilergebnisse verschieden, so ist dazu ein neuer Knoten für die umbenannte Top-Variable zu bauen, der die rekursiv ermittelten Teilresultate als Söhne besitzt. Dies entspricht der Kombination der Teilresultate, wie sie in Algorithmus 23 erfolgt. Damit entspricht das Vorgehen des Algorithmus bei der rekursiven Berechnung den in Satz 6.1 als korrekt bewiesenen Aufspaltungen.

6.3 Korrektheit der Terminalfälle

In diesem Abschnitt wird die Korrektheit der Terminalfälle, die im Algorithmus zur Berechnung des relationalen Produkts mit dem neuen Verfahren zur Berücksichtigung bereits berechneter Teilresultate aus Abschnitt 6.1 verwendet werden, erläutert. Die im Algorithmus *RelProductPartial()* benutzten Terminalfälle und deren Rückgabewerte sind in der in Algorithmus 24 in Abschnitt 6.1 angegebenen Funktion *TerminalCase()* aufgeführt. Die ersten beiden Terminalfälle treten auf, wenn der Wurzelknoten des BDDs zur Repräsentation der Transitionsrelation (R) oder der Wurzelknoten des BDDs zur Repräsentation der zu explorierenden Zustandsmenge (S) ein Terminalknoten für den Booleschen Wert 0 ist (siehe Zeile 2 in Algorithmus 24). Bei der kombinierten Berechnung des relationalen Produkts zwischen BDDs ergibt sich für diese Terminalfälle der Terminalknoten für den Booleschen Wert 0 als Rückgabewert (siehe Abschnitt 3.1). In einem Rekursionsschritt des Algorithmus mit dem neuen Verfahren zur Berücksichtigung bereits fertig berechneter Teilresultate ist zusätzlich noch das im BDD *Partial* enthaltene Teilresultat zu beachten. Dies ist gemäß der in einem Rekursionsschritt dieses Algorithmus durchzuführenden Berechnung $(\exists\vec{x}[R(\vec{x},\vec{x}') \wedge S(\vec{x})]\{\vec{x}' \leftarrow \vec{x}\} \vee Partial(\vec{x}))$ zu tun. Daher muss das Teilresultat und damit der BDD *Partial* mit der durch den Terminalknoten für den Booleschen Wert 0 repräsentierten leeren Menge vereinigt werden. Das Resultat daraus ist das Ergebnis der Berechnung des relationalen Produkts unter Berücksichtigung eines Teilresultats für einen solchen Rekursionsschritt. Aus dieser Vereinigung resultiert daher der BDD Partial als korrekter Rückgabewert für diese Terminalfälle.

Die anderen beiden im Algorithmus verwendeten Terminalfälle liegen vor, wenn der Wurzelknoten des BDDs R dem Terminalknoten für den Booleschen Wert 1 entspricht, oder wenn in einem Rekursionsschritt der BDD R und der BDD S gleich sind ($R == S$) (siehe Zeile 4 in Algorithmus 24). Diese Terminalfälle werden auch im Algorithmus zur kombinierten Berechnung des relationalen Produkts zwischen BDDs verwendet. Für

diesen Algorithmus wurde in Satz 3.3 (R ist Terminalknoten für den Booleschen Wert 1) und Satz 3.4 ($R == S$) in Abschnitt 3.3 die Korrektheit des Terminalknotens für den Booleschen Wert 1 als Rückgabewert dieser beiden Terminalfälle gezeigt. In beiden Sätzen sind zwei Fälle für das Resultat des jeweiligen Terminalfalls angegeben. Das Ergebnis des Terminalfalls, in dem der Wurzelknoten des BDDs R der Terminalknoten für den Wert 1 ist, ist der Boolesche Wert 0, falls der BDD zur Repräsentation einer Zustandsmenge dem Terminalknoten für den Booleschen Wert 0 entspricht. Ansonsten ist das Ergebnis der Boolesche Wert 1. Im Satz zum Terminalfall in dem die BDDs R und S gleich sind, wurde der Terminalknoten für den Booleschen Wert 0 als Rückgabewert bewiesen, falls der BDD R bzw. der BDD S dem Terminalknoten für den Booleschen Wert 0 entsprechen. Andernfalls ist der Rückgabewert dieses Terminalfalls ebenfalls der Terminalknoten für den Booleschen Wert 1.

In der in Algorithmus 24 für den Algorithmus *RelProductPartial()* angegebenen Funktion *TerminalCase()* wird als erstes geprüft, ob einer der Terminalfälle in denen der Wurzelknoten des BDDs R oder des BDDs S der Terminalknoten für den Wert 0 ist, vorliegt. Daher kann für die kombinierte Berechnung des relationalen Produkts ohne Berücksichtigung eines Teilresultats und die Terminalfälle in Zeile 4 der Funktion *TerminalCase()* nur noch der Terminalknoten für den Wert 1 als Rückgabewert auftreten. Ein noch zu berücksichtigendes und mit diesem Rückgabewert zu vereinigendes Teilresultat führt zu keiner Änderung des Rückgabewerts. Es sind bereits alle Eingabebelegungen, die im aktuellen BDD *Partial* zum Terminalknoten für den Wert 1 führen könnten, in der durch den Terminalknoten für den Wert 1 repräsentierten Menge enthalten. Daher entspricht der Terminalknoten für den Booleschen Wert 1 schon der Vereinigung des Ergebnisses des relationalen Produkts mit dem BDD *Partial* und der Terminalknoten für den Booleschen Wert 1 ist der korrekte Rückgabewert.

6.4 Möglichkeiten zur Bildberechnung

Für das in diesem Kapitel vorgestellte neue Verfahren zur Berücksichtigung bereits fertig berechneter Teilresultate werden im Folgenden mehrere Möglichkeiten zur Benutzung bei Bildberechnungen vorgestellt. Dies erfolgt hier für die Verwendung partitionierter Transitionsrelationen. Die präsentierten Ansätze können sowohl für durch BDDs als auch für durch TLEBDDs (siehe Algorithmus in Abschnitt 10.1 im Anhang) repräsentierte Mengen von Transitionen verwendet werden. In diesem Abschnitt werden die Ansätze für die Verwendung von BDDs vorgestellt. Außerdem können diese bis auf den in Algorithmus 27 angegebenen Ansatz auch bei Verwendung eines monolithischen BDDs zur Repräsentation einer Transitionsrelation benutzt werden. In diesem Fall werden lediglich nur Bildberechnungen mit der monolithischen Transitionsrelation durchgeführt. Die Verfahren werden nachfolgend für partitionierte Transitionsrelationen aus M Partitionen vorgestellt. Außerdem wird jeder Ansatz für eine komplette Explorationsrunde aus Bildberechnungen mit allen M Partitionen einer Transitionsrelation aufgeführt. Wie bei den für ETLEBDDs in der vorliegenden Arbeit präsentierten Explorationsstrategien werden dabei nur die für das jeweilige Verfahren zur Bildberechnung wichtigen Teile angegeben (siehe Abschnitt

5.8). Für die Durchführung einer Erreichbarkeitsanalyse müssten die im Folgenden für eine Explorationsrunde skizzierten Strategien solange ausgeführt werden, bis damit keine neuen Zustände mehr gefunden werden können.

Eine Explorationsrunde aus Bildberechnungen (Berechnungen des relationalen Produkts) mit jeder Partition einer partitionierten Transitionsrelation ist in Algorithmus 25 angegeben. Durch Aufruf der Funktion $Image(R_i, S_k)$ wird dort eine Bildberechnung, die bei Nachfolgerberechnungen zum Beispiel mit dem in Abschnitt 3.1 für die kombinierte Berechnung des relationalen Produkts mit einem BDD präsentierten Algorithmus erfolgen kann, durchgeführt. Als Übergabeparameter besitzt die Funktion einen BDD R_i, mit dem eine Partition der Transitionsrelation repräsentiert wird und einen BDD S_k, mit einer Menge zu explorierender Zustände. Nach jeder Bildberechnung werden die bei Bildberechnungen in vorherigen Explorationsrunden bereits gefundenen Zustände $S_{Reached}$ aus der Menge der durch eine Bildberechnung explorierten Zustände entfernt $(Image(R_i, S_k) \setminus S_{Reached})$. Anschließend wird die Menge der übrig bleibenden Zustände mit der Menge der bisher in der aktuellen Explorationsrunde neu gefundenen Zustände S_{k+1} vereinigt.

Algorithmus 25: Eine Explorationsrunde aus Bildberechnungen mit jeder Partition einer partitionierten Transitionsrelation mit der gewöhnlichen kombinierten Berechnung des relationalen Produkts (Ansatz BDD P. Imm).

```
1 foreach i ∈ M do
2 ⌊ S_{k+1} = S_{k+1} ∪ (Image(R_i, S_k) \ S_Reached);
```

Der erste Ansatz, der das neue Verfahren zur Berücksichtigung bereits fertig berechneter Teilresultate verwendet, ist in Algorithmus 26 dargestellt. Durch Aufruf der Funktion $ImageNew(R_i, S_k, \emptyset)$ wird dort eine Bildberechnung mit dem neuen Verfahren zur Berücksichtigung von Teilresultaten ausgeführt. Diese kann zum Beispiel mit dem in Abschnitt 6.1 für BDDs angegebenen Algorithmus erfolgen. Als bereits fertig berechnetes Teilresultat wird bei diesem Ansatz im ersten Aufruf des Algorithmus jeweils die leere Menge ($\emptyset$) übergeben. Dadurch wird das neue Verfahren zur Kombination von Teilresultaten nur für während der Bildberechnung neu auftretende Teilresultate verwendet. Ein Ansatz mit zusätzlicher Verwendung dieses Verfahrens zur Vereinigung des Resultats einer Bildberechnung mit den in einer Explorationsrunde bereits neu gefundenen Zuständen S_{k+1}, ist in Algorithmus 27 angegeben. Bei diesem wird die Zustandsmenge S_{k+1} bei Aufrufen des Algorithmus zur Bildberechnung (z.B. Aufruf des in Abschnitt 6.1 vorgestellten Algorithmus) als Übergabeparameter für ein zu berücksichtigendes Teilresultat verwendet $(ImageNew(R_i, S_k, S_{k+1}))$. Dadurch kann die explizite Vereinigung der Zustandsmenge S_{k+1} mit dem Ergebnis von Bildberechnungen vermieden werden.

Die in Abschnitt 8.5.1 angegebenen Ergebnisse von Verifikationsexperimenten zeigen, dass bei Verifikationsexperimenten jedes der drei bisher vorgestellten Verfahren die beste Laufzeit liefern kann. Deshalb wurde ein Ansatz zur Kombination dieser Verfahren entwickelt. Dieser versucht dynamisch das aktuell effizienteste Verfahren auszuwählen und

Algorithmus 26: Eine Explorationsrunde mit dem neuen Verfahren zur Berücksichtigung bereits fertig berechneter Teilresultate. Als Übergabeparameter für ein bereits fertig berechnetes Teilresultat wird die leere Menge verwendet (Ansatz BDD P. Part).

```
1 foreach i ∈ M do
2 ⌊ S_{k+1} = S_{k+1} ∪ (ImageNew(R_i, S_k, ∅) \ S_Reached);
```

Algorithmus 27: Eine Explorationsrunde mit dem neuen Verfahren zur Berücksichtigung bereits fertig berechneter Teilresultate. Als Übergabeparameter für ein bereits fertig berechnetes Teilresultat wird die Zustandsmenge S_{k+1} angegeben (Ansatz BDD P. Part All).

```
1 foreach i ∈ M do
2 ⌊ S_{k+1} = ImageNew(R_i, S_k, S_{k+1}) \ S_Reached;
```

ist in Algorithmus 28 dargestellt. Die Auswahl des für Bildberechnungen zu verwendenden Ansatzes wird durch Vergleich von für die verschiedenen Ansätze gespeicherten Ausführungszeiten durchgeführt. Dazu wird eine Variable *algType* benutzt, deren Wert das momentan für Bildberechnungen zu verwendende Verfahren anzeigt. Außerdem wird die Ausführungszeit der letzten Bildberechnung mit jedem Ansatz in einem Array *last* gespeichert. Bei den Variablenbelegungen der Variablen *algType* steht der Wert 0 für den Ansatz *BDD P. Imm* (siehe Algorithmus 25), der Wert 1 für den Ansatz *BDD P. Part* (siehe Algorithmus 26) und der Wert 2 für den Ansatz *BDD P. Part All* (siehe Algorithmus 27). Die Laufzeit einer Bildberechnung wird dabei durch Bilden der Differenz der nach und vor einer Bildberechnung gemessenen und gespeicherten CPU-Zeit bestimmt. Diese wird anschließend im entsprechenden Eintrag des Arrays *last* gespeichert (siehe Zeile 16 in Algorithmus 28). Änderungen der bei Bildberechnungen beteiligten BDDs können während der Durchmusterung des Zustandsraums, aufgrund von dadurch geänderten Eigenschaften für Bildberechnungen, zu großen Veränderungen der Laufzeit von Verfahren zur Bildberechnung führen. Im in Algorithmus 28 vorgestellten Ansatz wird dies zum einen durch Verringern der für gerade nicht für Bildberechnungen benutzten Verfahren gespeicherten Laufzeiten berücksichtigt. Dazu wird im Algorithmus eine Zählervariable *counter* benutzt, die die innerhalb eines Verfahrens zur Bildberechnung aufeinanderfolgend durchgeführten Bildberechnungen zählt. Nach 50 mit einem Verfahren hintereinander durchgeführten Bildberechnungen, werden die für die anderen beiden Ansätze zuletzt gemessenen und gespeicherten Laufzeiten verringert. Dazu werden deren zuletzt gemessene Laufzeiten mit dem Faktor 0.9 multipliziert. Damit sollen möglicherweise in der Zwischenzeit für aktuell auszuführende Bildberechnungen mit diesen Verfahren eingetretene Laufzeitverringerungen berücksichtigt werden.

Nach der Anpassung älterer gemessener Laufzeiten wird geprüft, ob ein Wechsel des momentan für Bildberechnungen verwendeten Ansatzes sinnvoll ist. Dazu wird die beim

aktuell verwendeten Verfahren benötigte Laufzeit mit den für die anderen beiden Ansätze gespeicherten letzten Ausführungszeiten verglichen (siehe Zeilen 21 und 24). Hier wird durch einen multiplikativen Faktor wieder berücksichtigt, dass die für die beiden anderen Verfahren zuvor gespeicherten Laufzeiten mittlerweile deutlich anders sein können. Ein Wechsel des zur Bildberechnung verwendeten Ansatzes wird durchgeführt, wenn die für die gerade ausgeführte Bildberechnung gemessene Laufzeit größer als die durch Multiplikation mit dem Faktor 0.8 angepassten gespeicherten Laufzeiten der anderen Verfahren ist.

Im hier vorgestellten einfachen dynamischen Algorithmus wurden die Anzahl der Bildberechnungen (50) und die verwendeten multiplikativen Faktoren (0.9 bzw. 0.8) für alle Testfälle fix gewählt. Dennoch konnten damit, wie die in Abschnitt 8.5.1 aufgeführten experimentellen Ergebnisse zeigen, für nahezu jeden Testfall die besten Laufzeiten unter allen in diesem Abschnitt vorgestellten Verfahren erzielt werden. Die Konstanten im Algorithmus wurden mit dem Ziel ausgewählt, dass der Algorithmus gut auf Änderungen der Eigenschaften für Bildberechnungen reagieren kann. Dennoch wurde versucht diese so zu wählen, dass ein momentan guter Ansatz zur Bildberechnung möglichst beibehalten wird. Bei der Auswahl des für Bildberechnungen zu verwendenden Ansatzes ist darauf zu achten, dass zu häufiges und auch fälschliches Wechseln des Ansatzes auch zu einer Erhöhung der Laufzeit führen kann. Der Algorithmus wurde als Veranschaulichung des Prinzips eines Algorithmus entwickelt, der sich dynamisch an Veränderungen der Effizienz von verschiedenen Verfahren zur Bildberechnung anpassen kann. Eine Erweiterung des hier präsentierten Algorithmus wäre die Anpassung und Optimierung der fest gewählten Parameter während der Durchmusterung des Zustandsraums. Dies wäre zum Beispiel durch den Einsatz von Lernverfahren möglich.

Algorithmus 28: Verfahren, das versucht durch Laufzeitvergleiche aus den drei in diesem Abschnitt präsentierten Ansätzen zur Durchführung von Bildberechnungen dynamisch das aktuell effizienteste Verfahren auszuwählen (Ansatz BDD P. ALG COMB).

```
1  foreach i ∈ M do
2      start = time(); counter++;
3      if algType == 0 then
4          S_{k+1} = S_{k+1} ∪ (Image(R_i, S_k) \ S_Reached);
5          // Approach: BDD P. Imm
6          end = time(); IndexOne=1; IndexTwo=2;
7      else
8          if algType == 1 then
9              S_{k+1} = S_{k+1} ∪ (ImageNew(R_i, S_k, ∅) \ S_Reached);
10             // Approach: BDD P. Part Image
11             end = time(); IndexOne=0; IndexTwo=2;
12         else
13             S_{k+1} = ImageNew(R_i, S_k, S_{k+1}) \ S_Reached;
14             // Approach: BDD P. Part All
15             end = time(); IndexOne=0; IndexTwo=1;
16     last[algType] = end - start;
17     if counter > 50 then
18         last[IndexOne] = last[IndexOne] · 0.9;
19         last[IndexTwo] = last[IndexTwo] · 0.9;
20         counter = 0;
21     if (last[algType] ≥ 0.8 · last[IndexOne] && (last[IndexOne] ≤
22         last[IndexTwo])) then
23         algType = IndexOne; counter = 0;
24     else if last[algType] ≥ 0.8 · last[IndexTwo] then
25         algType = IndexTwo; counter = 0;
```

6.5 Abgrenzung zu verwandten Arbeiten

Der in diesem Kapitel vorgestellte Algorithmus zur kombinierten Berechnung des relationalen Produkts benutzt ein neues Verfahren zur Berücksichtigung bereits fertig berechneter Teilresultate. Für den in Kapitel 3 zur kombinierten Berechnung des relationalen Produkts mit BDDs eingeführten Algorithmus wurde eine Abgrenzung zu verwandten Arbeiten in Abschnitt 3.5 angegeben. Die dort für die Merkmale der kombinierten Berechnung des relationalen Produkts angegebene Abgrenzung bezüglich der in Abschnitt 2.5 präsentierten verwandten Arbeiten gilt auch für den in diesem Kapitel vorgestellten Algorithmus. Dies ist der Fall, da hier ebenfalls eine kombinierte Berechnung des relationalen Produkts durchgeführt wird. Von der in Kapitel 3 für BDDs vorgestellten kombinierten Berechnung des relationalen Produkts, unterscheidet sich der Algorithmus durch das zusätzlich verwendete neue Verfahren zur Berücksichtigung bereits fertig berechneter Teilresultate und die sich dadurch ergebenden Anpassungen des Algorithmus. Durch simultanes Durchlaufen eines übergebenen BDDs für ein Teilresultat, kann die explizite Vereinigung dieses mit dem Resultat noch ausstehender Berechnungen vermieden werden. Verwandte Arbeiten, in denen ebenfalls die Kombination von fertig berechneten Teilresultaten mit dem Ergebnis noch ausstehender Berechnungen auf diese Weise durchgeführt wird, konnten keine gefunden werden. Zusätzlich wurden in diesem Kapitel noch verschiedene Strategien zur Benutzung des hier neu vorgestellten Algorithmus präsentiert. Diese wurden für eine Breitensuche zur Durchführung der Erreichbarkeitsanalyse angegeben. Mehrere Ansätze, die davon Abweichen und eine Kombination aus Breiten- und Tiefensuche ausführen, wurden in Abschnitt 2.5.7 aufgeführt.

7 Verbesserte dynamische Symmetriereduktion

Symmetriereduktionstechniken wurden für die Modellprüfung entwickelt, um den bei ihr benötigten Speicherbedarf zu verringern. Dies wird durch Beschränkung der Durchmusterung des Zustandsraums auf Repräsentanten von Äquivalenzklassen von unter vorhandenen Symmetrien äquivalenten Zuständen erreicht. Außerdem kann die Anwendung von Symmetriereduktionstechniken zu Laufzeitverbesserungen führen. In Abschnitt 2.4 der vorliegenden Arbeit wurde die Ausnutzung von Symmetrien bei der Modellprüfung eingeführt. Dabei wurden sowohl für die explizite, als auch für die symbolische Modellprüfung mit BDDs, Symmetriereduktionstechniken vorgestellt. Ein Verfahren zur Symmetriereduktion bei der symbolischen Modellprüfung mit BDDs, ist die dynamische Symmetriereduktion. Diese wurde in Abschnitt 2.4.2.3 beschrieben. In diesem Kapitel werden Verbesserungen zur Benutzung der dynamischen symbolischen Symmetriereduktion vorgestellt.

7.1 Dynamische Symmetriereduktion für partitionierte Transitionsrelationen

Die für die symbolische Modellprüfung mit BDDs entwickelte dynamische Symmetriereduktion wurde in Abschnitt 2.4.2.3 bei Benutzung eines monolithischen BDDs zur Repräsentation der Transitionsrelation eingeführt. Sie kann aber auch bei Verwendung partitionierter Transitionsrelationen eingesetzt werden. Bei Anwendung der dynamischen Symmetriereduktion werden Repräsentanten von Äquivalenzklassen von unter Symmetrien äquivalenten Zuständen nach Bildberechnungen dynamisch berechnet. Dazu werden die mit einer Bildberechnung explorierten und in einem BDD gespeicherten Zustände unmittelbar nach der Bildberechnung auf Repräsentanten der entsprechenden Äquivalenzklassen abgebildet. In Algorithmus 25 in Abschnitt 6.4 wurde für partitionierte Transitionsrelationen eine aus einer Bildberechnung mit jeder Partition einer Transitionsrelation bestehende Explorationsrunde angegeben. Bei der Erreichbarkeitsanalyse mit dem damit veranschaulichten Ansatz werden solange solche Explorationsrunden aus Bildberechnungen ausgeführt, bis damit keine noch nicht entdeckten Zustände mehr gefunden werden können. Dieser Ansatz, bei dem eine Breitensuche zur Exploration des Zustandsraums ausgeführt wird, entspricht einem naheliegenden und häufig verwendeten Ansatz zur Erreichbarkeitsanalyse mit disjunktiv partitionierten Transitionsrelationen. Ein naiver Ansatz zur Integration der dynamischen Symmetriereduktion in dieses Vorgehen, ist die dynamische Kanonisierung der durch eine Bildberechnung explorierten Zustände nach jeder Bildberechnung mit einer Partition der Transitionsrelation.

In diesem Abschnitt wird eine neue Vorgehensweise zur Benutzung der dynamischen Symmetriereduktion bei der Erreichbarkeitsanalyse mit partitionierten Transitionsrelationen vorgestellt. Bei dieser wird versucht die Eigenschaften der dynamischen Symmetriereduktion gezielt zur Erzielung von Laufzeitverbesserungen auszunutzen. Dies geschieht durch Veränderung der Reihenfolge der Ausführung von Bildberechnungen mit Partitionen der Transitionsrelation und damit der Strategie zur Exploration des Zustandsraums. Die dazu neu entwickelte Vorgehensweise ist in Algorithmus 29 dargestellt. Sie kann mit partitionierten Transitionsrelationen aus BDDs oder TLEBDDs verwendet werden. In diesem Abschnitt wird der neue Ansatz für die Benutzung von partitionierten Transitionsrelationen aus BDDs vorgestellt. Der Algorithmus führt keine Breitensuche durch, sondern es werden bei der Exploration des Zustandsraums mit dem hier vorgestellten Algorithmus immer aufeinanderfolgend Bildberechnungen mit der gleichen Partition der Transitionsrelation durchgeführt, solange damit noch nicht entdeckte und damit mit jeder Partition der Transitionsrelation weiterzuexplorierende Zustände gefunden werden können. Dadurch mit einer Partition der Transitionsrelation neu gefundene Zustandsmengen können anschließend bei Bildberechnungen mit anderen Partitionen der Transitionsrelation auf einmal mit einer Bildberechnung weiterexploriert werden. Dies führt häufig zu größeren nach Bildberechnungen zu kanonisierenden Zustandsmengen, wodurch die Repräsentantenberechnung bei der dynamischen Symmetriereduktion mit größeren Zustandsmengen durchgeführt werden kann. Der Algorithmus kann sowohl für die Vorwärtserreichbarkeitsanalyse, mit wiederholt ausgeführten Nachfolgerberechnungen, sowie bei entsprechenden Anpassungen auch für die Rückwärtserreichbarkeitsanalyse, bei der wiederholt Vorgängerberechnungen durchgeführt werden (siehe Abschnitt 2.1.6.3), benutzt werden. Dazu ist je nach Explorationsrichtung unter anderem die Berechnung des relationalen Produkts, die durch Aufruf der Funktion *Image*(R_i, *ExploredStates*) in Zeile 15 im Algorithmus erfolgt, entsprechend auszuführen. Als Übergabeparameter besitzt diese Funktion den BDD einer Partition der Transitionsrelation (R_i) und eine Menge von zu explorierenden Zuständen (*ExploredStates*). Nachfolgend wird die neue Vorgehensweise für die Vorwärtserreichbarkeitsanalyse vorgestellt. Bei den dort auszuführenden Nachfolgerberechnungen kann für die Funktion *Image()* zum Beispiel der in Abschnitt 3.1 eingeführte Algorithmus zur kombinierten Berechnung des relationalen Produkts für die Nachfolgerberechnung mit BDDs benutzt werden. Der in diesem Abschnitt präsentierte Algorithmus wird hier für komponentenweise partitionierte Transitionsrelationen aus einer Partition für jede Komponente vorgestellt. Diese enthält jeweils alle in der Transitionsrelation vorhandenen Transitionen der Komponente. Für die Verwendung anderer Partitionierungen der Transitionsrelation ist der Algorithmus entsprechend anzupassen.

In Algorithmus 29 werden zuerst Initialisierungen von Variablen vorgenommen. Dazu wird in Zeile 1 des Algorithmus ein BDD *Init* mit der Menge der Anfangszustände S_0 des betrachteten asynchronen nebenläufigen Systems initialisiert. Anschließend werden diese mit der Abstraktionsfunktion α der dynamischen Symmetriereduktion (siehe Abschnitt 2.4.2.3) auf die Menge der entsprechenden kanonischen Repräsentanten abgebildet. Dadurch werden nachfolgend nur Bildberechnungen mit den Repräsentanten zu den Anfangszuständen ausgeführt. Dies hat außerdem zur Folge, dass keine Repräsentan-

Algorithmus 29: Neuer Ansatz zur Erreichbarkeitsanalyse mit einer partitionierten Transitionsrelation, bei Benutzung der dynamischen symbolischen Symmetriereduktion.

```
1  BDD Init = S_0; Init = α(Init);
2  BDD Reached = Init; BDD ExploredStates = ∅; BDD NewExplored = ∅;
3  BDD ToExplore[NumberOfComponents];
4  Bool Finish = false;
5  for i = NumberOfComponents−1; i ≥ 0; i−− do
6      ToExplore[i] = Init;
7  /* Here we integrated the use of state symmetries (see Algorithmus 30 and section
   7.3) for the initial states
8  for i = NumberOfComponents−1; i > 0; i−− do
9      <Insertion of Algorithmus 30>
10 */
11 while Finish == false do
12     for i = NumberOfComponents−1; i ≥ 0; i−− do
13         ExploredStates = ToExplore[i];
14         while ExploredStates ≠ ∅ do
15             ExploredStates = Image(R_i, ExploredStates);
16             ExploredStates = α(ExploredStates) & !NewExplored & !Reached;
17             NewExplored | = ExploredStates;
18         Reached | = NewExplored;
19         foreach k ∈ Components do
20             if k ≠ i then
21                 ToExplore[k] | = NewExplored;
22             else
23                 ToExplore[k] = ∅;
24         NewExplored = ∅;
25         /* Here we integrated the use of state symmetries (see Algorithmus 30 and
           section 7.3) for explored states
26         <Insertion of Algorithmus 30> */
27     Finish = true;
28     foreach i ∈ Components do
29         if ToExplore[i] ≠ ∅ then
30             Finish = false;
```

tenberechnungen für aus unreduzierten Anfangszuständen explorierte Zustände durchgeführt werden müssen. Aufgrund der vorhandenen Symmetrien würden damit keine anderen Repräsentanten, als die aus Bildberechnungen mit den symmetriereduzierten Anfangszuständen resultierenden Repräsentanten, ermittelt werden. Nach der Kanonisierung der Anfangszustände werden ein BDD zur Speicherung aller während der Erreichbarkeitsanalyse gefundenen Zustände (*Reached*), ein BDD in dem die durch Bildberechnungen neu gefundenen Repräsentanten von Äquivalenzklassen von Zuständen gespeichert werden (*ExploredStates*) und ein BDD zur Speicherung aller bei erschöpfender Exploration des Algorithmus für eine Komponente (mit der entsprechenden Partition der Transitionsrelation) neu gefundenen Zustände (*NewExplored*) angelegt. Der BDD *Reached* wird dabei mit den zuvor ermittelten symmetriereduzierten Anfangszuständen initialisiert. Beim zu Beginn dieses Abschnitts beschriebenen Vorgehen zur Benutzung einer partitionierten Transitionsrelation (siehe auch Algorithmus 25), erfolgt in einer Explorationsrunde eine Bildberechnung mit jeder Partition der Transitionsrelation. Dabei werden die Bildberechnungen mit den verschiedenen Partitionen der Transitionsrelation direkt nacheinander und mit der gleichen zu explorierenden Zustandsmenge durchgeführt. Wie bereits beschrieben, werden im hier vorgestellten Verfahren stattdessen mit einem BDD für eine Partition der Transitionsrelation solange Bildberechnungen ausgeführt, wie damit bisher noch nicht gefundene Repräsentanten von Zuständen ermittelt werden können. Da neu explorierte Zustände dabei direkt weiter exploriert werden, können sich die Zustandsmengen, für die für verschiedene Partitionen der Transitionsrelation Bildberechnungen durchzuführen sind, unterscheiden. Aus diesem Grund wird in Algorithmus 29 für jede Partition der Transitionsrelation ein eigener BDD zur Speicherung der bei der nächsten Bildberechnung mit dieser Partition zu explorierenden Zustände benutzt. Dazu wird in Zeile 3 des Algorithmus ein Array *ToExplore* mit einem BDD für jede Komponente (damit hier für jede Partition der Transitionsrelation) angelegt. Diese BDDs werden in Zeile 6 für jede Komponente mit der Menge der zu Beginn zu explorierenden symmetriereduzierten Anfangszustände initialisiert.

Das Explorieren aller von den Anfangszuständen erreichbaren Zustände wird von einer *while*-Schleife, die in Zeile 11 von Algorithmus 29 angegeben ist, gesteuert. Die Schleife wird solange ausgeführt, bis alle erreichbaren Zustände gefunden wurden. Innerhalb der *while*-Schleife werden Bildberechnungen mit jeder Partition der Transitionsrelation durchgeführt. Da der Algorithmus für eine komponentenweise partitionierte Transitionsrelation aus einer Partition für jede Komponente vorgestellt wird, wird dazu im Algorithmus in einer *for*-Schleife über alle Komponenten iteriert (Zeile 12). In dieser Schleife wird die Menge der mit einer Partition der Transitionsrelation zuerst zu explorierenden Zustände *ExploredStates* mit den für diese Partition im Array *ToExplore* vorhandenen Zuständen initialisiert (siehe Zeile 13). Anschließend werden in einer weiteren *while*-Schleife solange Bildberechnungen mit dieser Partition der Transitionsrelation ausgeführt, bis dadurch keine bisher noch unentdeckten Repräsentanten von Zuständen mehr gefunden werden können (siehe Zeilen 14 bis 17). Die in einer solchen Explorationsrunde neu gefundenen Zustände werden im BDD *NewExplored* angesammelt. Dazu wird in jeder Iteration der *while*-Schleife zuerst durch Aufruf der Funktion *Image()* eine Bildberechnung durchge-

führt (Zeile 15). Anschließend werden die symmetriereduzierten Repräsentanten zu den mit der Bildberechnung explorierten Zuständen ermittelt. Dazu wird die Abstraktionsfunktion α der dynamischen Symmetriereduktion verwendet (Zeile 16). Aus der resultierenden Zustandsmenge werden dort außerdem die aktuell mit den BDDs *NewExplored* und *Reached* repräsentierten Zustandsmengen entfernt. Die Menge *NewExplored* enthält die innerhalb der aktuellen Ausführung der inneren *while*-Schleife neu gefunden Zustände und die Menge *Reached* die insgesamt bereits gefundenen Zustände. Daher sind im BDD *ExploredStates*, nach Entfernen der in diesen Mengen enthaltenen Zustände, nur noch während der Erreichbarkeitsanalyse bisher noch nicht gefundene Zustände enthalten. Diese Zustände werden danach in Zeile 17 zur Menge *NewExplored* hinzugefügt. Dadurch werden die gerade neu explorierten Repräsentanten im BDD für die Menge der Zustände, die in der aktuellen Ausführung der inneren *while*-Schleife neu gefunden wurden, abgespeichert. Nach Auffinden aller Zustände, die ausgehend von einer Menge von für eine Partition der Transitionsrelation zu explorierenden Zuständen gefunden werden können, werden diese zur Menge der insgesamt bereits gefundenen Zustände *Reached* hinzugefügt (Zeile 18). Anschließend werden die Zustandsmengen, in denen die Zustände mit denen für die einzelnen Komponenten noch Bildberechnungen ausgeführt werden müssen angesammelt werden, aktualisiert. Dazu werden die neu gefundenen und im BDD *NewExplored* gespeicherten Zustände für alle Komponenten, bis auf die Komponente für die gerade in der *while*-Schleife Bildberechnungen durchgeführt wurden, zur bereits im entsprechenden Eintrag des Arrays *ToExplore* gespeicherten Zustandsmenge hinzugefügt (Zeilen 19 bis 23). Dadurch können die bei Bildberechnungen für eine Komponente neu gefundenen Zustände von jeder Komponente weiterexploriert werden. Für die Komponente, für die gerade Bildberechnungen ausgeführt wurden, wird der Eintrag des Arrays *ToExplore* auf die leere Menge gesetzt. Die im BDD *NewExplored* enthaltenen Zustände wurden für diese zuvor in der *while*-Schleife bereits weiterexploriert.

Nach der erschöpfenden Durchführung von Bildberechnungen für alle Partitionen der Transitionsrelation wird im Algorithmus geprüft, ob bereits alle von den Anfangszuständen erreichbaren Zustände gefunden wurden. Ist dies der Fall, kann der Algorithmus terminiert werden. Dazu wird überprüft, ob es eine Partition der Transitionsrelation gibt, in deren zugeordnetem BDD *ToExplore* noch zu explorierende Zustände enthalten sind (Zeilen 28 bis 30). Falls für keine Partition mehr Zustände zu explorieren sind, besitzt die Boolesche Variable *Finish* den Wert *true*, auf den sie in Zeile 27 vor dieser Überprüfung gesetzt wird. Der Wert der Variable wird nicht geändert, wenn es keine Partition gibt, für die noch Zustände zu explorieren sind. Damit zeigt der Wert *true* dieser Variablen an, dass der Algorithmus terminiert werden kann. In Abschnitt 7.2 wird gezeigt, dass die Erreichbarkeitsanalyse und damit das Auffinden aller von den Anfangszuständen erreichbaren Zuständen durch die im Algorithmus verwendete Vorgehensweise korrekt ausgeführt wird.

Bei Ausführen von Bildberechnungen für eine Komponente, werden in Algorithmus 29 immer erschöpfend Bildberechnungen durchgeführt. Dadurch und weil zwischen zwei erschöpfenden Ausführungen von Bildberechnungen für eine Komponente für alle anderen Komponenten Bildberechnungen ausgeführt werden, können sich in den BDDs *ToExplore*

sehr große Zustandsmengen ansammeln. Erfolgt eine Bildberechnung mit einer größeren Menge *ToExplore*, kann mit dieser eine größere Menge an Zuständen exploriert werden. Auch innerhalb der erschöpfenden Bildberechnungen für eine Komponente können mit Bildberechnungen größere Zustandsmengen ermittelt werden, wenn die vorausgehende Bildberechnung eine größere zu explorierende Zustandsmenge liefert. Bei Verwendung der dynamischen Symmetriereduktion, kann die dort auszuführende Repräsentantenberechnung damit für größere Zustandsmengen gleichzeitig erfolgen. Außerdem werden diese Zustandsmengen von der Komponente, für die gerade erschöpfend Bildberechnungen ausgeführt werden, direkt weiter exploriert. Für bei der Repräsentantenberechnung für mehrere Zustände gleich anfallende Berechnungen (Vertauschungen von Variablen, siehe Abschnitt 2.4.2.3), müssen diese nur einmal ausgeführt werden. Dadurch eingesparte Berechnungen führen zur Einsparung der dafür benötigten Laufzeit. Durch die Repräsentantenberechnung für größere Zustandsmengen beim hier vorgestellten Verfahren, kann die Häufigkeit des Auftretens von für mehrere Zustände dabei gleich durchzuführenden Berechnungen gesteigert werden.

Im in diesem Abschnitt vorgestellten Algorithmus werden Bildberechnungen immer ausgehend von symmetriereduzierten Repräsentanten von Zuständen durchgeführt. Bei asynchronen nebenläufigen Systemen mit Transitionslokalität und der Verwendung von partitionierten Transitionsrelationen, in denen die Partitionen der Transitionsrelation jeweils nur Transitionen einer Komponente beinhalten, kann eine weitere Eigenschaft beobachtet werden. Dort treten bei Bildberechnungen mit einer Partition der Transitionsrelation im Vergleich zu den Ausgangszuständen sogar nur Änderungen des globalen Zustands und des lokalen Zustands der Komponente, für die in der Partition Transitionen gespeichert sind, auf. Dadurch sind bei der Repräsentantenberechnung für neu explorierte Zustände notwendige Vertauschungen von Variablen ausgehend von den gleichen Teilen von Zuständen auszuführen. Es müssen dabei bei der Repräsentantenberechnung für mehrere explorierte Zustände anfallende gleiche Berechnungen nur einmal ausgeführt werden. Durch den Start mit Vertauschungen von Variablen bei den gleichen Teilen von Zuständen, können bei Verwendung partitionierter Transitionsrelationen durch vorhandene gleiche Berechnungen resultierende Einsparungen effizienter ausgenutzt werden, als bei monolithischen Transitionsrelationen. Dies, die größeren Zustandsmengen und die direkte Weiterexploration neu gefundener Zustände für Partitionen der Transitionsrelation sind Faktoren für die mit dem hier präsentierten Algorithmus erzielten Laufzeitverbesserungen (siehe Ergebnisse Verifikationsexperimente in Abschnitt 8.6).

7.2 Korrektheit des Algorithmus

Ziel der Erreichbarkeitsanalyse ist es, alle von einer gegebenen Zustandsmenge erreichbaren Zustände zu ermitteln. Im letzten Abschnitt wurde ein Algorithmus für die Vorwärtserreichbarkeitsanalyse präsentiert, mit dem alle von den Anfangszuständen eines Systems erreichbaren Zustände exploriert werden können. Hier wird im Folgenden gezeigt, dass die in diesem Algorithmus dazu verwendete Vorgehensweise korrekt ist. Der Algorithmus wurde in Abschnitt 7.1 für eine komponentenweise partitionierte Transitionsrelation, in der

für jede Komponente eines Systems eine Partition mit allen Transitionen der Komponente vorhanden ist, angegeben. Für eine solche Partitionierung der Transitionsrelation wird die Korrektheit des Algorithmus nachfolgend gezeigt. Die dazu angegebenen Beweise können aber für die Benutzung anderer Partitionierungen der Transitionsrelation angepasst werden. Nachfolgend wird in Lemma 7.1 zuerst gezeigt, dass die kanonischen Repräsentanten der Anfangszustände und alle Zustände, die während der Erreichbarkeitsanalyse in Algorithmus 29 neu gefunden werden, mit allen Partitionen der Transitionsrelation weiterexploriert werden.

Lemma 7.1. *Gegeben sei ein asynchrones nebenläufiges System $M^m = (S, S_0, R)$ aus m (NumberOfComponents) replizierten Komponenten mit $m \geq 1$ und eine komponentenweise partitionierte Transitionsrelation, in der für jede Komponente des Systems eine Partition mit allen Transitionen der Komponente vorhanden ist. Seien außerdem Symmetrien durch die m replizierten Komponenten vorhanden (siehe Abschnitt 2.4.1.1). Dann werden bei Ausführung von Algorithmus 29 die symmetriereduzierten Repräsentanten der Anfangszustände ($\alpha(S_0)$) und alle während dem Ablauf des Algorithmus neu gefundenen Repräsentanten von Äquivalenzklassen aus unter den vorhandenen Symmetrien äquivalenten Zuständen mit allen m (NumberOfComponents) vorhandenen Partitionen der Transitionsrelation weiterexploriert.*

Beweis. Zum Beweis dieses Lemmas werden zwei Fälle unterschieden:

1. Fall: Zustand s ist ein symmetriereduzierter Repräsentant eines Anfangszustands des Systems ($s \in \alpha(S_0)$)

   Dann wird wegen *Finish=false* in Zeile 4 des Algorithmus, die in Zeile 11 beginnende *while*-Schleife ausgeführt. Es gilt für alle Komponenten K_i, mit $0 \leq i \leq$ *NumberOfComponents*-1:

   - nach Ausführung von Zeile 6 des Algorithmus ist $s \in$ *ToExplore[i]* und
   - $s \in$ *ExploredStates* in der ersten Iteration der ersten Ausführung der in Zeile 14 beginnenden *while*-Schleife für Komponente K_i.
   - Damit wird s in Zeile 15 für Komponente K_i weiterexploriert.

2. Fall: Der Repräsentant Zustand s wird bei einer Bildberechnung für eine Komponente K_i ($0 \leq i \leq$ *NumberOfComponents*-1) und der anschließenden Repräsentantenberechnung neu gefunden

   Hier lassen sich wieder zwei Fälle unterscheiden:

   a) Weiterexploration für Komponente K_i

      Repräsentant s wurde nach Bildberechnung für Komponente K_i neu gefunden
      $\Rightarrow$ Zustand s in Iteration der *while*-Schleife in der er neu gefunden wird in Zeile

16 des Algorithmus in *ExploredStates*
⇒ Abbruchbedingung *while*-Schleife in Zeile 14 nicht erfüllt
⇒ in der nächsten Iteration dieser *while*-Schleife wird in Zeile 15 eine Bildberechnung mit Zustand s für K_i durchgeführt.

b) Weiterexploration für eine beliebige andere Komponente K_j, mit $0 \leq j \leq$ *NumberOfComponents*-1, und $j \neq i$

Repräsentant s wurde nach Bildberechnung für Komponente K_i neu gefunden
⇒ Zustand s in Iteration der *while*-Schleife (aus Zeile 14) in der er gefunden wird in Zeile 17 des Algorithmus in *NewExplored*
⇒ nach Beendigung Bildberechnungen für Komponente K_i wird s in Zeile 21 zu *ToExplore* von K_j hinzugefügt, wodurch gilt *ToExplore[j]* $\neq \emptyset$. Damit ist der Zustand s bei der nächsten Bildberechnung für K_j in der zu explorierenden Zustandsmenge enthalten
⇒ es werden noch Bildberechnungen für K_j durchgeführt, da entweder

- K_j in der aktuellen *for*-Schleife in der über die Komponenten iteriert wird (Zeile 12) noch nicht an der Reihe war, oder
- die Boolesche Variable *Finish* auf den Wert *false* gesetzt wird (Zeile 30), da gilt *ToExplore[j]* $\neq \emptyset$. In diesem Fall werden in der nächsten Ausführung der *for*-Schleife Bildberechnungen für K_j durchgeführt.

Damit werden für alle NumberOfComponents Komponenten Bildberechnungen mit den Repräsentanten der Anfangszustände und mit allen während dem Ablauf des Algorithmus neu gefundenen Repräsentanten von Äquivalenzklassen aus unter den vorhandenen Symmetrien äquivalenten Zuständen durchgeführt. □

Unter Benutzung von Lemma 7.1 wird anschließend im Beweis zu Satz 7.2 gezeigt, dass mit Algorithmus 29 alle von den Anfangszuständen einer Quotienten Kripke-Struktur erreichbaren symmetriereduzierten Zustände gefunden werden.

Satz 7.2. *Gegeben sei ein asynchrones nebenläufiges System $M^m = (S, S_0, R)$ aus m (NumberOfComponents) replizierten Komponenten mit $m \geq 1$ und eine komponentenweise partitionierte Transitionsrelation, in der für jede Komponente des Systems eine Partition mit allen Transitionen der Komponente vorhanden ist. Seien außerdem Symmetrien durch die m replizierten Komponenten vorhanden (siehe Abschnitt 2.4.1.1).*

Dann wird mit Algorithmus 29 die Menge aller von den symmetriereduzierten Anfangszuständen ($\alpha(S_0)$) erreichbaren symmetriereduzierten Zustände des Systems ($\alpha(S)$) exploriert.

Beweis. Die Gültigkeit des Satzes kann direkt aus Lemma 7.1 abgeleitet werden. Sei $s \in (\alpha(S) \ / \ \alpha(S_0))$ ein beliebiger von den symmetriereduzierten Anfangszuständen des Systems erreichbarer Repräsentant. Wir nehmen an, dass dieser Repräsentant nicht bei Ausführung von Algorithmus 29 bei mit den symmetriereduzierten Anfangszuständen

beginnender Zustandsraumexploration gefunden werden kann. Da der Zustand von den symmetriereduzierten Anfangszuständen des Systems erreichbar ist, kann er durch wiederholt ausgeführte Bildberechnungen mit der gesamten Transitionsrelation und nachfolgender Symmetriereduktion, beginnend mit den symmetriereduzierten Anfangszuständen, erreicht werden. Wird der Zustand von Algorithmus 29 nicht gefunden, dann muss es einen letzten gefundenen symmetriereduzierten Vorgängerzustand von s oder einen symmetriereduzierten Anfangszustand geben, für den ein durch Anwendung der gesamten Transitionsrelation erreichbarer symmetriereduzierter Zustand von Algorithmus 29 nicht gefunden wird. Dazu müssten Zustände von Algorithmus 29 nicht mit der gesamten Transitionsrelation weiterexploriert werden. Dies ist ein Widerspruch zu Lemma 7.1, in dem gezeigt wurde, dass die symmetriereduzierten Anfangszustände und alle während der Durchmusterung des symmetriereduzierten Zustandsraums mit Algorithmus 29 neu gefundenen symmetriereduzierten Zustände vom Algorithmus mit der für jede Komponente vorhandenen Partition der Transitionsrelation weiterexploriert werden. □

7.3 Implementierung von Zustandssymmetrien

Bei der Modellprüfung mit Benutzung von Symmetriereduktionstechniken für Systeme mit Symmetrien durch replizierte Komponenten kann auch eine weitere Art von Symmetrien, sogenannte Zustandssymmetrien, ausgenutzt werden. Dies sind Symmetrien, die in einzelnen Systemzuständen vorhanden sind. Sie resultieren aus äquivalenten Eigenschaften von Komponentenzuständen in einem Systemzustand. Zur Ausnutzung von Zustandssymmetrien für einen Zustand, wird für diesen eine Zustandssymmetriepartition über den im System vorhandenen Komponenten gebildet. In jeder Partition einer Zustandssymmetriepartition sind nur Komponenten mit bezüglich des Systemzustands identischen Eigenschaften enthalten. Für die Komponenten in einer Partition einer Zustandssymmetriepartition müssen Transitionen nur für eine repräsentative Komponente berücksichtigt werden. Der Grund dafür ist, dass sich für alle Komponenten einer Zustandssymmetriepartition nach Anwenden der Transitionen auf den Systemzustand für den die Zustandssymmetriepartition ermittelt wurde und nach der anschließenden Repräsentantenberechnung, die gleichen symmetriereduzierten Repräsentanten ergeben würden. Dadurch können durch die Berücksichtigung von Zustandssymmetrien redundante Repräsentantenberechnungen vermieden werden. Ausführlicher eingeführt wurden Zustandssymmetrien in Abschnitt 2.4.2.4 der vorliegenden Arbeit.

Bisher wurden nach unserem Kenntnisstand nur Verfahren zur Ausnutzung von Zustandssymmetrien bei der expliziten Modellprüfung entwickelt. Vor allem bei Systemen mit vielen replizierten Komponenten konnten dort durch ihre Ausnutzung große Laufzeitverbesserungen erzielt werden. In diesem Abschnitt wird ein neuer Ansatz zur Ausnutzung von Zustandssymmetrien bei der dynamischen symbolischen Symmetriereduktion präsentiert. Im Unterschied zur expliziten Modellprüfung, bei der die Berechnung und Ausnutzung von Zustandssymmetrien für einzelne Zustände erfolgt, wird bei der symbolischen Modellprüfung mit BDDs mit Mengen von Zuständen und Transitionen gearbeitet. Die Berechnung von vorhandenen Zustandssymmetrien muss daher hier für Mengen von

Zuständen durchgeführt werden. Der in diesem Abschnitt vorgestellte Ansatz zur Ausnutzung von Zustandssymmetrien wurde zur Benutzung bei vorhandener vollständiger Symmetrie (siehe Abschnitt 2.4.1.2) entworfen. Zustandssymmetrien können aber auch bei anderen vorliegenden Symmetriegruppen, wie zum Beispiel der Rotationssymmetrie, verwendet werden. Allerdings ist bei der Ausnutzung von Zustandssymmetrien darauf zu achten, dass die Zustandssymmetrien mit wenig Zeitaufwand berechnet werden können. Dann können durch ihre Ausnutzung möglichst Laufzeitgewinne erzielt werden. Die Ermittlung von Zustandssymmetrien kann in Abhängigkeit der vorhandenen Symmetriegruppe unterschiedlichen Aufwand und damit unterschiedlichen Zeitbedarf erfordern. Im schlimmsten Fall kann deren Ausnutzung sogar zu einer Laufzeiterhöhung führen.

Algorithmus 30: Ausnutzung von Zustandssymmetrien beim in Algorithmus 29 in Abschnitt 7.1 präsentierten Algorithmus zur Vorwärtserreichbarkeitsanalyse mit Benutzung der dynamischen Symmetriereduktion.

```
1 stateSymmStates = ∅;
2 //i is the currently active component
3 if i ≠ 0 then
4 |   foreach globalIdstVar do
5 |   |   stateSymmStates | = equal(globalIdstVar, i);
6 |   |   stateSymmStates | = equal(globalIdstVar, i-1);
7 |   stateSymmStates = ToExplore[i-1] & !stateSymmStates;
8 |   stateSymmStates = stateSymmStates & equal(i, i-1);
9 |   ToExplore[i-1] = ToExplore[i-1] & !stateSymmStates;
```

Die im Folgenden vorgestellte Implementierung der Erkennung und Ausnutzung von Zustandssymmetrien ist in Algorithmus 30 angegeben. Durch Integration dieser in den in Algorithmus 29 in Abschnitt 7.1 präsentierten Ansatz zur Vorwärtserreichbarkeitsanalyse mit Benutzung der dynamischen Symmetriereduktion, können bei diesem zusätzlich Zustandssymmetrien ausgenutzt werden. Dazu ist die in Algorithmus 30 dargestellte Implementierung in den Zeilen 9 und 26 in Algorithmus 29 einzufügen (diese Zeilen sind in Algorithmus 29 entsprechend gekennzeichnet). Wie oben beschrieben, müssen bei vorhandenen Zustandssymmetrien und einer Zustandssymmetriepartition für einen Systemzustand, für jede Partition der Zustandssymmetriepartition nur Transitionen einer in der Partition enthaltenen Komponente ausgeführt werden. Um mit der Ausnutzung von Zustandssymmetrien möglichst große Laufzeitgewinne erzielen zu können, ist es wichtig diese schnell entdecken zu können. Zusätzlich sollten möglichst viele vorhandene Zustandssymmetrien ausgenutzt werden können. Dies war das Ziel der in Algorithmus 30 angegebenen Implementierung der Ausnutzung von Zustandssymmetrien und der Integration dieser in Algorithmus 29 zur Vorwärtserreichbarkeitsanalyse. In Algorithmus 29 wird die Ausnutzung von Zustandssymmetrien für die symmetriereduzierten Anfangszustände eines Systems in Zeile 9 durchgeführt. Für während der Durchmusterung des Zustandsraums neu gefundene Zustände wurde die Ausnutzung von Zustandssymmetrien in Zeile 26 des

Algorithmus integriert. Die Ausnutzung von Zustandssymmetrien wird in diesem Abschnitt für eine komponentenweise partitionierte Transitionsrelation, mit einer Partition der Transitionsrelation für jede Komponente, angegeben. Für diese Art der Repräsentation der Transitionsrelation wurde auch Algorithmus 29 angegeben. Bei Verwendung mehrerer Partitionen mit Transitionen einer Komponente ist der hier beschriebene Ansatz entsprechend anzupassen.

In Algorithmus 29 werden in einer aus Bildberechnungen für alle Komponenten bestehenden Explorationsrunde zuerst Bildberechnungen für die Komponente mit dem höchsten Komponentenindex durchgeführt (siehe Zeile 12 in Algorithmus 29). Nach erschöpfender Ausführung von Bildberechnungen für eine Komponente, erfolgen dort immer Bildberechnungen für die Komponente mit dem nächstniedrigeren Komponentenindex. Dies ermöglicht die hier vorgestellte Implementierung der Ausnutzung von Zustandssymmetrien und deren Berechnung immer zwischen benachbarten Komponenten (Komponentenindizes i und $i-1$). Die Berechnung und Berücksichtigung vorhandener Zustandssymmetrien wird während der Durchmusterung des Zustandsraums immer nach der erschöpfenden Ausführung von Bildberechnungen für Komponenten durchgeführt. Nach erschöpfenden Bildberechnungen für eine Komponente mit dem Index i, werden in Algorithmus 29 die damit neu gefundenen Zustände zu den Mengen *ToExplore* aller anderen Komponenten hinzugefügt (siehe Zeile 21 in Algorithmus 29). Dadurch können diese Zustände später für diese Komponenten weiterexploriert werden. Bei der in Algorithmus 30 implementierten Ausnutzung von Zustandssymmetrien, werden Zustände mit zwischen benachbarten Komponenten vorhandenen Zustandssymmetrien anschließend aus der Menge *ToExplore* für die benachbarte Komponente mit dem niedrigeren Komponentenindex ($i-1$ in Algorithmus 30) entfernt. Die entfernten Zustände werden dadurch bei den nachfolgend für diese Komponente ausgeführten Bildberechnungen nicht mehr berücksichtigt. Bei Berücksichtigung dieser bei der nächsten Bildberechnung, würden damit nach Symmetriereduktion nur bereits gefundene symmetriereduzierte Repräsentanten erneut ermittelt werden. Durch ihre Entfernung aus der Menge *ToExplore*, kann der nur für deren Berechnung notwendige Aufwand bei der Repräsentantenberechnung vermieden werden.

Die in der vorliegenden Arbeit zur Ermittlung von Zustandssymmetrien zwischen zwei benachbarten Komponenten (mit Komponentenindizes i und $i-1$) verwendete Implementierung ist in Algorithmus 30 dargestellt. Dort wird zuerst ein BDD *stateSymmStates* angelegt (Zeile 1 in Algorithmus 30). In diesem werden Zustände mit Zustandssymmetrien zwischen den benachbarten Komponenten, die in der Menge *ToExplore[i-1]* für die Komponente mit dem Index $i-1$ enthalten sind, gespeichert (siehe Zeilen 4 bis 8 in Algorithmus 30). Für Zustände mit Zustandssymmetrien müssen von mindestens einer Komponente jeder Zustandssymmetriepartition Bildberechnungen ausgeführt werden. Bei der hier präsentierten Ausnutzung von Zustandssymmetrien werden für die Anfangszustände und auch nach erschöpfenden Bildberechnungen für alle Komponentenindizes bis auf die Komponente mit dem Index 0 Zustandssymmetrien berechnet. Dadurch werden aus der Menge *ToExplore* für die Komponente mit dem höchsten Komponentenindex keine Zustände mit Zustandssymmetrien entfernt. Damit erfolgen für diese Komponente für alle zu explorierenden Zustände Bildberechnungen. Dies führt zum Beispiel für die Anfangszustände

dazu, dass diese Komponente bei Anfangszuständen mit Zustandssymmetrien zwischen allen Komponenten, die Komponente der Zustandssymmetriepartition ist, die als Repräsentant immer Transitionen ausführt. Bei der hier vorgestellten Implementierung werden Zustandssymmetrien zwischen benachbarten Komponenten ausgenutzt, wenn keine der vorhandenen globalen id-sensitiven Variablen auf eine der benachbarten Komponenten zeigt. Wenn dies für einen Zustand der Fall wäre, würde die Eigenschaft der Äquivalenz der beiden Komponenten für einen Systemzustand (siehe Definition 2.33 in Abschnitt 2.4.2.4) nicht mehr gelten, da eine globale id-sensitive Variable nur auf eine Komponente zeigen kann. Um dies bei der Ermittlung von Zustandssymmetrien zu berücksichtigen, wird für jede globale id-sensitive Variable für beide benachbarten Komponenten ein BDD gebaut. Diese repräsentieren jeweils die Menge aller Eingabebelegungen, in denen die aktuell betrachtete globale id-sensitive Variable auf die Komponente mit dem Index i (siehe Zeile 5) bzw. die Komponente mit dem Index $i-1$ (siehe Zeile 6) zeigt. Der Bau dieser BDDs wird durch die Funktion *equal* durchgeführt, die als Übergabeparameter die aktuell betrachtete globale id-sensitive Variable und einen Komponentenindex besitzt. Die dafür erzeugten BDDs werden im BDD *stateSymmStates* vereinigt.

Nach Erzeugung dieser BDDs für alle globalen id-sensitiven Variablen, werden die dadurch im BDD *stateSymmStates* gespeicherten Zustände von der Menge *ToExplore* der von der Komponente mit dem Index $i-1$ noch zu explorierenden Zustände abgezogen (siehe Zeile 7). Dadurch werden diese bei der nachfolgenden Berechnung der Menge der Zustände mit Zustandssymmetrien nicht mehr berücksichtigt. Anschließend wird in Zeile 8 ein BDD erzeugt, der die Menge aller Eingabebelegungen mit gleichen lokalen Zuständen der Komponenten mit den Indizes i und $i-1$ repräsentiert. Dieser wird danach mit dem BDD *stateSymmStates* verundet. Der Bau des BDDs mit den Eingabebelegungen mit gleichen lokalen Zuständen wird wieder durch die Operation *equal* durchgeführt. Diese bekommt hier als Parameter die beiden Komponentenindizes übergeben, für die die lokalen Zustände gleich sein sollen. Nach diesem Schritt enthält die Menge *stateSymmStates* die Menge der Zustände aus *ToExplore[i-1]*, in denen keine globale id-sensitive Variable auf eine der benachbarten Komponenten zeigt und in denen die Komponenten im gleichen lokalen Zustand sind. Damit besteht diese Menge aus Zuständen der Menge *ToExplore[i-1]*, mit Zustandssymmetrien zwischen den benachbarten Komponenten. Diese Zustände werden anschließend aus der Menge *ToExplore[i-1]*, die die für die Komponente mit dem Index $i-1$ noch zu explorierenden Zustände enthält, entfernt (siehe Zeile 9). Dadurch werden diese Zustände bei der nachfolgend mit den Zuständen aus *ToExplore[i-1]* für diese Komponente ausgeführten Bildberechnung nicht mehr berücksichtigt. Aufgrund der in diesen Zuständen vorhandenen Zustandssymmetrien würden bei Weiterexploration dieser Zustände für die Komponente mit dem Index $i-1$, nur bereits durch Bildberechnungen für Komponenten mit höheren Komponentenindizes bereits gefundene Repräsentanten erneut exploriert werden. Durch die Vermeidung der Exploration dieser Zustände, sind die dafür zur Ermittlung der Repräsentanten notwendigen Berechnungen nicht notwendig. Im Gegensatz zur expliziten Modellprüfung, wird bei der symbolischen Modellprüfung mit Mengen von Zuständen gearbeitet. Bei der dynamischen symbolischen Symmetriereduktion werden Mengen von Repräsentanten von Zustandsmengen gemeinsam berechnet.

Daher können zur Repräsentantenberechnung für einen Zustand benötigte Berechnungen nur eingespart werden, wenn in der insgesamt nach einer Bildberechnung zu kanonisierenden Zustandsmenge kein Zustand enthalten ist, für dessen Kanonisierung eine solche Berechnung nötig ist. Die in der vorliegenden Arbeit für die Ausnutzung von Zustandssymmetrien präsentierten experimentellen Ergebnisse (siehe Abschnitt 8.6) zeigen aber, dass durch die Benutzung der hier vorgestellten Implementierung von Zustandssymmetrien dennoch bei einigen Testfällen große Laufzeitverbesserungen erzielt werden können.

Mit der in diesem Abschnitt vorgestellten Implementierung von Zustandssymmetrien werden nicht alle vorhandenen Zustandssymmetrien ausgenutzt. Es erfolgt während der erschöpfenden Exploration von Zuständen für eine Komponente keine Berücksichtigung von Zustandssymmetrien. Zustandssymmetrien werden stattdessen erst nach der erschöpfenden Durchführung von Bildberechnungen für eine Komponente berechnet. Außerdem werden sie immer nur zwischen einer Komponente und einer benachbarten Komponente, für die als nächstes Bildberechnungen ausgeführt werden, berechnet. Während der Entwicklung der hier präsentierten Implementierung der Ausnutzung von Zustandssymmetrien wurden auch Implementierungen, bei denen Zustandssymmetrien nach jeder Bildberechnung und direkt zwischen allen Komponenten berücksichtigt wurden, getestet. Bei diesem Ansatz fällt ein größerer Zeitaufwand zum Ermitteln von Zustandssymmetrien an und in den Verifikationsexperimenten wurden dafür im Vergleich mit dem hier vorgestellten Verfahren schlechtere Laufzeitwerte beobachtet. Dies zeigt, dass die effiziente Ausnutzung vorhandener Zustandssymmetrien wichtig ist. Dabei kann es vorteilhaft sein, nicht alle vorhandenen Zustandssymmetrien zu berechnen und auszunutzen.

7.4 Korrektheit Benutzung von Zustandssymmetrien

Das Ausnutzen von Zustandssymmetrien erfolgt korrekt, wenn immer von jeder Partition einer Zustandssymmetriepartition eines Zustands mindestens eine repräsentative Komponente Transitionen ausführt. Für die in Abschnitt 7.3 beschriebene Implementierung bedeutet dies, dass für jeden aus der Menge *ToExplore* einer Komponente aufgrund von Zustandssymmetrien entfernten Zustand, mindestens eine andere Komponente der entsprechenden Zustandssymmetriepartition Transitionen ausführen muss. Für die Anfangszustände wurde die in Algorithmus 30 beschriebene Ausnutzung von Zustandssymmetrien in Algorithmus 29 (wie dort gekennzeichnet) in Zeile 9 eingefügt. Wird dort ein Zustand aus der Menge *ToExplore* für eine Komponente mit einem Index $i - 1$ entfernt, dann ist die Komponente mit dem Index i aufgrund der dazu in Algorithmus 30 durchgeführten Berechnungen in der gleichen Zustandssymmetriepartition des Zustands. Für die Komponente mit dem Index 0 findet keine Berechnung von Zustandssymmetrien statt (siehe Zeile 3 in Algorithmus 30) und es werden für die Komponente mit dem höchsten Index keine Zustände aus *ToExplore* entfernt. Daher gibt es zu jedem für eine Komponente aus der Menge der zu explorierenden Zustände aufgrund von Zustandssymmetrien entfernten Anfangszustand eine Komponente aus der gleichen Zustandssymmetriepartition mit einem höheren Komponentenindex, die Transitionen für diesen Anfangszustand ausführt. Dies ist die Komponente mit dem höchsten Komponentenindex aus einer Partition einer

Zustandssymmetriepartition, da für diese keine benachbarte Komponente mit dem nächsthöheren Index aus der gleichen Zustandssymmetriepartition vorhanden ist oder es ist die Komponente mit dem höchsten Index (*NumberOfComponents-1*). Da für jeden aus *ToExplore* einer Komponente aufgrund von Zustandssymmetrien aus den Anfangszuständen entfernten Zustand der Zustand in *ToExplore* (den Anfangszuständen) für eine andere Komponente enthalten ist, findet für diese andere Komponente mindestens ein Durchlauf der in Zeile 14 in Algorithmus 29 aufgeführten *while*-Schleife statt (*Finish==false* und *ExploredStates*$\neq \emptyset$) und es werden für diesen Zustand damit für eine Komponente der Zustandssymmetriepartition Bildberechnungen ausgeführt. Nachfolgend wird in Satz 7.3 gezeigt, dass auch für nach Bildberechnungen für eine Komponente aus der Menge *ToExplore* der benachbarten Komponente aufgrund von Zustandssymmetrien mit Algorithmus 30 entfernte Zustände (siehe Zeile 26 in Algorithmus 29) Bildberechnungen für eine Komponente ausgeführt werden.

Satz 7.3. *Gegeben sei ein asynchrones nebenläufiges System* $M^m = (S, S_0, R)$ *aus m (NumberOfComponents) replizierten Komponenten mit* $m \geq 1$ *und eine komponentenweise partitionierte Transitionsrelation, in der für jede Komponente eine Partition mit allen Transitionen der Komponente vorhanden ist. Sei außerdem vollständige Symmetrie zwischen den m replizierten Komponenten des Systems vorhanden.*

Dann gilt, wenn in Algorithmus 29 unter Benutzung des in Algorithmus 30 angegebenen Ansatzes ein Zustand aufgrund von Zustandssymmetrien aus der Menge ToExplore[i-1] nach Bildberechnungen für eine Komponente mit einem Index i ($0 < i \leq NumberOfComponents - 1$) *entfernt wird, dass Bildberechnungen mit diesem Zustand für eine Komponente mit einem Index j* ($i \leq j \leq NumberOfComponents - 1$) *aus der Partition der Zustandssymmetriepartition dieses Zustands mit der Komponente mit dem Index* $i - 1$ *ausgeführt werden.*

Beweis. Sei Zustand s ein beliebiger während der Erreichbarkeitsanalyse für das gegebene System mit Algorithmus 29 durch Repräsentantenberechnung nach einer Bildberechnung auftretender unter den vorhandenen Symmetrien kanonischer Repräsentant. Seien außerdem zwei beliebige in diesem Zustand benachbarte Komponenten mit den Indizes $i - 1$ und i, die in der gleichen Partition der Zustandssymmetriepartition dieses Zustands sind, gegeben. Der Zustand s werde nun, bei Verwendung der in Abschnitt 7.3 angegebenen Ausnutzung von Zustandssymmetrien, aus der Menge der für die Komponente mit dem Index $i - 1$ zu explorierenden Zustände *ToExplore[i-1]* entfernt. Dies geschieht nach der erschöpfenden Durchführung von Bildberechnungen für die Komponente mit dem Index i. Dann zeigt keine globale id-sensitive Variable auf eine dieser beiden Komponenten und der lokale Zustand der beiden Komponenten ist gleich (siehe Algorithmus 30). Weil Zustand s in der Menge *ToExplore[i-1]* für die Komponente mit dem Index $i - 1$ enthalten ist, kann er nicht bei Bildberechnungen mit dieser Komponente neu exploriert worden sein. Ansonsten würde er durch die *while*-Schleife in Zeile 15 direkt weiterexploriert werden. Es lassen sich damit für die weiteren mit Zustand s ausgeführten Bildberechnungen zwei Fälle unterscheiden (nachfolgend angegebene Zeilenangaben beziehen sich auf Algorithmus 29):

- 1. Fall: Zustand s wurde bei den für die Komponente mit dem Index i vorangehend erschöpfend ausgeführten Bildberechnungen neu gefunden

  ⇒ Zustand s ist symmetriereduzierter Repräsentant eines bei einer Bildberechnung für die Komponente mit dem Index i in Zeile 15 von Algorithmus 29 explorierten Zustands
  ⇒ Der symmetriereduzierte Zustand s wird in Zeile 16 nach der in Zeile 15 ausgeführten Bildberechnung ermittelt und zur Menge *ExploredStates* hinzugefügt
  ⇒ Damit gilt anschließend $ExploredStates \neq \emptyset$ (Zeile 14) und es werden weiter Bildberechnungen für die Komponente mit dem Index i ausgeführt
  ⇒ Bei der nachfolgenden Bildberechnung in der *while*-Schleife wird eine Bildberechnung für Zustand s für die Komponente mit dem Index i durchgeführt ($s \in$ *ExploredStates*, siehe Zeile 15).

- 2. Fall: Zustand s wurde bei einer vorangehenden Bildberechnung für eine andere Komponente als die Komponente mit dem Index i neu gefunden. Dann wurde er dort nach den erschöpfend ausgeführten Bildberechnungen zu den Mengen *ToExplore[i]* und *ToExplore[i-1]* für die Komponenten mit den Indizes i und $i-1$ hinzugefügt (Zeile 21). Hier sind wieder zwei Fälle zu unterscheiden:

  - 1. Fall: Zustand s wurde vor der nächsten Ausführung von Bildberechnungen für die Komponente mit dem Index i nicht aus *ToExplore[i]* entfernt

    ⇒ Zustand s ist in der Menge der Zustände für die als nächstes für die Komponente mit dem Index i Bildberechnungen ausgeführt werden enthalten (siehe Zeile 13)
    ⇒ Bei der nächsten Ausführung von Bildberechnungen für die Komponente mit dem Index i, wird in Zeile 15 eine Bildberechnung mit Zustand s ausgeführt.

  - 2. Fall: Zustand s wurde vor der nächsten Ausführung von Bildberechnungen für die Komponente mit dem Index i, aufgrund von Zustandssymmetrien zur benachbarten Komponente mit dem Index $i+1$, aus *ToExplore[i]* entfernt

    Wie in Abschnitt 7.3 beschrieben, werden beim in der vorliegenden Arbeit zur Ausnutzung von Zustandssymmetrien verwendeten Ansatz keine Zustände aus der Menge *ToExplore[NumberOfComponents-1]* für die Komponente mit dem höchsten Komponentenindex entfernt.
    Die Entfernung eines Zustands s aus der Menge *ToExplore[j]* für eine Komponente mit einem Index j, mit $i < j < NumberOfComponents - 1$, wird nur durchgeführt, wenn nach der in Algorithmus 30 angegebenen Implementierung eine Zustandssymmetrie zur Komponente mit dem nächsthöheren Index besteht. Da aus *ToExplore[NumberOfComponents-1]* keine Zustände entfernt

werden, gibt es eine Komponente mit einem Index j, die für den Zustand s in der gleichen Partition der Zustandssymmetriepartition wie die Komponente mit dem Index $i-1$ ist und die unter diesen Komponenten den größten Komponentenindex aus der Menge $\{i+1, i+2, i+3, ..., NumberOfComponents-1\}$ besitzt. Für diese Komponente ist s entweder vor einer Ausführung erschöpfender Bildberechnungen in *ToExplore[j]* enthalten oder Zustand s wurde durch die Komponente mit dem Index j neu gefunden. In beiden Fällen führt die Komponente mit dem Index j Bildberechnungen für Zustand s aus.

Damit wird der Zustand s in jedem Fall von einer Komponente aus der Partition der Zustandssymmetriepartition dieses Zustands, in der sich auch Komponente $i-1$ befindet, weiterexploriert. □

7.5 Abgrenzung zu verwandten Arbeiten

In diesem Kapitel wurde ein neuer Ansatz zur Verwendung der dynamischen symbolischen Symmetriereduktion bei der Modellprüfung mit partitionierten Transitionsrelationen vorgestellt. Außerdem wurde ein Verfahren zur zusätzlichen Ausnutzung von Zustandssymmetrien präsentiert. Die dynamische Symmetriereduktion wurde in Abschnitt 2.4.2.3 der vorliegenden Arbeit für die symbolische Modellprüfung mit BDDs eingeführt. Im Vergleich zur Symmetriereduktion bei der expliziten Modellprüfung, bei der Repräsentanten von Äquivalenzklassen von unter Symmetrien äquivalenten Zuständen für einzelne Zustände berechnet werden, wird bei der symbolischen BDD-basierten Modellprüfung mit Mengen von Zuständen gearbeitet. Daher müssen dort Mengen von Repräsentanten zu Zustandsmengen gemeinsam berechnet werden. Mehrere Verfahren dazu, die alle bestimmte Nachteile besitzen, wurden in Abschnitt 2.4.2.2 vorgestellt. Die dynamische Symmetriereduktion ist ein Ansatz, bei dem der Bau des häufig intraktabel großen BDDs der Orbit-Relation zur Abbildung von Zuständen auf ihre Repräsentanten nicht nötig ist. Repräsentanten zu durch Bildberechnungen ermittelten Zustandsmengen, werden bei ihr dynamisch nach den Bildberechnungen durch Anwendung einer Abstraktionsfunktion berechnet. Eine Abgrenzung der dynamischen Symmetriereduktion von den anderen in dieser Arbeit für die symbolische Modellprüfung vorgestellten Verfahren zur Symmetriereduktion, ist am Anfang von Abschnitt 2.4.2.3 angegeben.

In der ursprünglichen Version wurde die dynamische Symmetriereduktion wie in Abschnitt 2.4.2.3 beschrieben, für die Benutzung mit einer monolithischen Transitionsrelation eingeführt. Der in der vorliegenden Arbeit präsentierte neue Ansatz wurde für die Benutzung der dynamischen Symmetriereduktion mit partitionierten Transitionsrelationen entwickelt. Dabei wurde versucht Anwendungen der Abstraktionsfunktion zur Repräsentantenberechnung, die die Laufzeit der dynamischen Symmetriereduktion bestimmen, effizienter durchzuführen. Repräsentantenberechnungen werden dort immer nach Bildberechnungen für nur eine Partition der Transitionsrelation durchgeführt. Außerdem erfolgen erschöpfend Bildberechnungen mit einer Partition der Transitionsrelation, bis damit keine noch nicht gefundenen neuen Zustände mehr entdeckt werden können. Erst danach werden Bildberechnungen mit anderen Partitionen der Transitionsrelation ausgeführt. Die

während der erschöpfenden Ausführung von Bildberechnungen neu explorierten Zustände werden angesammelt. Vor Ausführung von Bildberechnungen für andere Komponenten werden sie zu den für diese zu explorierenden Zustandsmengen hinzugefügt. Damit können nachfolgende Bildberechnungen für andere Komponenten und die anschließenden Kanonisierungen der dabei explorierten Zustände für größere Zustandsmengen als bei der gewöhnlichen Verwendung von partitionierten Transitionsrelationen ausgeführt werden. Dadurch müssen für mehrere Zustände bei einer Repräsentantenberechnung gleich anfallende Berechnungen nur einmal durchgeführt werden. Außerdem führt die Verwendung von komponentenweise partitionierten Transitionsrelationen für asynchrone nebenläufige Systeme mit Transitionslokalität bei Bildberechnungen mit Partitionen der Transitionsrelation zur Änderung der gleichen Teile von Zuständen (siehe letzter Absatz in Abschnitt 7.1), was ebenfalls vorteilhaft für die Repräsentantenberechnung ist. Wie die experimentellen Ergebnisse in Abschnitt 8.6 zeigen, können mit dem neuen Ansatz im Vergleich mit der gewöhnlichen Benutzung der dynamischen Symmetriereduktion für eine monolithische Transitionsrelation, oder der naiven Verwendung der dynamischen Symmetriereduktion mit partitionierten Transitionsrelationen (skizziert im 1. Absatz von Abschnitt 7.1), große Laufzeitverbesserungen resultieren.

Das erschöpfende Ausführen von Bildberechnungen für Partitionen der Transitionsrelation erfolgt im in Abschnitt 7.1 vorgestellten Algorithmus mit dem Ziel, möglichst große Zustandsmengen für die nachfolgenden Bildberechnungen mit anderen Partitionen der Transitionsrelation anzusammeln. In Abschnitt 2.5.7 wurden weitere Ansätze, bei denen ebenfalls erschöpfend Bildberechnungen mit Partitionen der Transitionsrelation vor Bildberechnungen mit anderen Partitionen durchgeführt werden, vorgestellt. Bei keinem dieser Ansätze wurden zusätzlich Verfahren zur Symmetriereduktion verwendet. In den dort beschriebenen Ansätzen aus [39] und [96] werden ebenfalls wiederholt Fixpunkte durch Bildberechnungen für Teile der Transitionsrelation berechnet. Durch das damit erfolgende Abweichen von der Breitensuche zur Exploration des Zustandsraums wurde dort versucht, die bei der Breitensuche oft anfallenden großen BDDs für Zwischenergebnisse kleiner zu halten. In [133] wurde ein Ansatz, bei dem eine Kombination von Fixpunktberechnungen durch Bildberechnungen mit Transitionen einzelner Komponenten eines Systems und Bildberechnungen mit Transitionen aller Komponenten verwendet wird, eingeführt. Ziel dieses Ansatzes war die Implementierung der Reduktion des Zustandsraums durch Halbordnung ([8, 56, 112, 157]) für die BDD-basierte Modellprüfung. Für MDDs wurde der *Saturation*-Algorithmus [46, 47, 48, 49, 151, 194] entwickelt. Bei diesem werden, wie in Abschnitt 2.5.7 beschrieben, Knoten von MDDs durch wiederholte Fixpunktberechnungen von unten nach oben in Entscheidungsdiagrammen wiederholt saturiert. Ziel dieses Vorgehens ist eine speichereffizientere und schnellere Exploration des Zustandsraums. Ein *Chaining* genanntes Verfahren, bei dem bei Bildberechnungen mit einer Partition der Transitionsrelation neu gefundene Zustände bei nachfolgenden Bildberechnungen für andere Partitionen direkt weiterexploriert werden, wurde in [169, 179] vorgestellt. Bei der traditionellen Breitensuche mit einer partitionierten Transitionsrelation werden in einer vollständigen Bildberechnung für alle Partitionen Bildberechnungen mit der gleichen Zustandsmenge ausgeführt. Durch das direkte Weiterexplorieren neu gefundener Zustände

beim *Chaining* ergibt sich ebenfalls ein Abweichen von der traditionellen Breitensuche, was zur Erzielung von Laufzeitverbesserungen entwickelt wurde. Das direkte Weiterexplorieren neu gefundener Zustände bei nachfolgenden Bildberechnungen mit anderen Partitionen der Transitionsrelation, erfolgt auch im in Abschnitt 7.1 eingeführten Algorithmus.

Zustandssymmetrien wurden in Abschnitt 2.4.2.4 dieser Arbeit eingeführt. Bisher wurden für die Ausnutzung von Zustandssymmetrien nach unserem Kenntnisstand nur Ansätze für die explizite Modellprüfung entwickelt. Bei den in Abschnitt 2.4.2.4 dazu präsentierten und in den expliziten Modellprüfern SMC [178] und Murphi [7] implementierten Verfahren, werden Zustandssymmetrien für einzelne Zustände berechnet. Werden dort in einem Zustand Zustandssymmetrien entdeckt, wird die Ausführbarkeit von Transitionen auf einen repräsentativen Prozess jeder Zustandssymmetriepartition beschränkt. Das in diesem Kapitel zur Ausnutzung von Zustandssymmetrien bei der BDD-basierten symbolischen Modellprüfung vorgestellte Verfahren ermöglicht die simultane Berechnung und Ausnutzung von Zustandssymmetrien für Zustandsmengen. Dabei werden Zustandssymmetrien ausgenutzt, in dem Zustände mit Zustandssymmetrien bei komponentenweise partitionierten Transitionsrelationen aus der Menge der für Komponenten weiter zu explorierenden Zustände entfernt werden. Dadurch werden diese anschließend bei Bildberechnungen nicht berücksichtigt und die redundante Repräsentantenberechnung für die aus solchen Zuständen explorierten Zustände kann vermieden werden.

8 Experimentelle Ergebnisse

In diesem Kapitel werden Ergebnisse von Verifikationsexperimenten mit den in den vorangehenden Kapiteln beschriebenen Ansätzen präsentiert. Außerdem wird deren Leistungsfähigkeit verglichen. Es werden experimentelle Ergebnisse zum Bau der Transitionsrelation und zur Vorwärtserreichbarkeitsanalyse angegeben. Die dazu durchgeführten Verifikationsexperimente erfolgten mit einer um die für die Verifikationsexperimente nötigen Techniken erweiterten Version des Modellprüfers Sviss (siehe Abschnitt 2.4.3.3). Dieser benutzt für Operationen mit BDDs das BDD-Programmpaket CUDD (siehe Abschnitt 2.2.3). Ausgeführt wurden die Verifikationsexperimente auf einem Rechner mit einer Intel Core i7 CPU mit 3.4 GHz und 16 GB Hauptspeicher. Da der Modellprüfer Sviss nur einen Kern eines Mehrkernprozessors nutzt, wurde für die Experimente immer nur ein Kern des Prozessors verwendet.

8.1 Verwendete Testfälle

In diesem Abschnitt werden die Testfälle beschrieben, mit denen Verifikationsexperimente durchgeführt wurden. Bei allen Testfällen handelt es sich um asynchrone nebenläufige Systeme (siehe Abschnitt 2.3) aus mehreren replizierten Komponenten. Der erste in den in diesem Kapitel angegebenen Tabellen mit experimentellen Ergebnissen aufgeführte Testfall wird in den Tabellen *Mut* genannt. Dieser ist eine Implementierung eines Protokolls, das den wechselseitigen Ausschluss für einen kritischen Abschnitt in einem System mit mehreren nebenläufigen Komponenten sicherstellt. Eine Erklärung und ein Zustandsübergangsdiagramm des Protokolls ist in Abschnitt 2.1.2 der vorliegenden Arbeit zu finden. In diesem Protokoll besitzt jede Komponente nur drei lokale Zustände, von denen ein lokaler Zustand dem kritischen Abschnitt entspricht. Außerdem gibt es eine globale id-sensitive Variable (siehe Abschnitt 2.4.1.1), die auf die Komponente zeigt, der gerade der Zutritt in den kritischen Abschnitt erlaubt ist. Dieser Testfall wurde ausgewählt, da er das Durchführen von Verifikationsexperimenten mit einer sehr großen Anzahl an Komponenten ermöglicht. Bei den Verifikationsexperimenten zur Vorwärtserreichbarkeitsanalyse mit diesem Testfall wurde dabei als Eigenschaft die Invariante überprüft, dass keine zwei Komponenten gleichzeitig im kritischen Abschnitt sind. Aufgrund der Regelung des Zutritts zum kritischen Abschnitt in diesem Protokoll, ist diese Eigenschaft für diesen Testfall immer erfüllt.

Der zweite für Verifikationsexperimente verwendete Testfall wird in den Tabellen mit den experimentellen Ergebnissen *MutL* bezeichnet. Dieser Testfall ist eine erweiterte Version des oben beschriebenen einfachen Protokolls zur Sicherstellung des wechselseitigen Ausschlusses, bei dem eine größere Anzahl an globalen Zuständen und lokalen Zuständen

der Komponenten vorhanden ist. Jede Komponente besitzt hier ebenfalls einen lokalen Zustand, der einem kritischen Abschnitt entspricht. Der Zutritt zum kritischen Abschnitt wird wieder über eine globale id-sensitive Variable geregelt und nur die Komponente, auf die die id-sensitive Variable gerade zeigt, darf ihren kritischen Abschnitt betreten. Neben der globalen id-sensitiven Variablen gibt es in diesem Testfall noch weitere globale Variablen. Es wird für jeden möglichen lokalen Kontrollzustand der Komponenten eine globale Variable verwendet, deren Wert die Anzahl der Komponenten angibt, die sich in einem Systemzustand im entsprechenden Kontrollzustand befinden. Außerdem besitzt eine Komponente inklusive des kritischen Abschnitts statt drei Kontrollzuständen, wie eine Komponente im davor vorgestellten Protokoll, zehn lokale Kontrollzustände. Zusätzlich hat jede Komponente eine weitere lokale Variable. In dieser wird bei einem Wechsel des Kontrollzustands einer Komponente die Anzahl der Komponenten gespeichert, deren lokaler Zustand im aktuellen Systemzustand dem neuen Kontrollzustand der Komponente entspricht. Diese Information kann durch die globalen Variablen, in denen die Anzahl der Komponenten die gerade in den jeweiligen Kontrollzuständen sind gespeichert ist, gewonnen werden. Als Eigenschaft wird bei den Verifikationsexperimenten zur Vorwärtserreichbarkeitsanalyse hier ebenfalls überprüft, dass keine zwei Komponenten gleichzeitig im lokalen Zustand für ihren kritischen Abschnitt sind.

Der Peterson-Algorithmus [158] (abgekürzt mit *Pet* in den Tabellen mit den experimentellen Ergebnissen) ist ein Protokoll zur Sicherstellung des wechselseitigen Ausschlusses. In einem System aus n-Komponenten wird hier einer Komponente der Zutritt zum kritischen Abschnitt durch eine Serie von $n-1$ Wettstreiten gewährt. In jedem Wettstreit gibt es mindestens einen Verlierer. Das Protokoll erfüllt die Eigenschaft des wechselseitigen Ausschlusses, da genau eine Komponente den letzten Wettstreit gewinnt. In diesem Testfall gibt es mehrere globale Variablen und als Eigenschaft wurde in den Verifikationsexperimenten geprüft, dass keine zwei Komponenten gleichzeitig in ihrem kritischen Abschnitt sind. Als weiterer Testfall wurde das Philosophenproblem (engl. dining philosophers problem), das in den nachfolgend aufgeführten Tabellen mit *DP* abgekürzt wird, verwendet. Die für die Verifikationsexperimente verwendete Implementierung ist eine Nachahmung der Monitor-Lösung aus [18]. Bei dieser können Philosophen nur beginnen zu essen, wenn sie die zwei Gabeln die sie zum Essen benötigen gleichzeitig aufnehmen können. Dadurch kann das Auftreten eines Deadlocks vermieden werden. Als Eigenschaft wurde hier überprüft, das keine zwei benachbarten Philosophen gleichzeitig essen können. Diese Eigenschaft wird ebenfalls durch die gleichzeitige Aufnahme beider benötigter Gabeln durch die Philosophen sichergestellt. Ein von S. German entwickeltes Cache-Kohärenz-Protokoll ist der in den Tabellen mit *CCP* bezeichnete Testfall (siehe zum Beispiel [161]). In diesem Protokoll können Client-Komponenten geteilten oder exklusiven Zugriff auf eine globale geteilte Cachezeile anfordern. Der Cache-Zugriff wird dabei durch eine zentrale sogenannte Home-Komponente koordiniert. In der in dieser Arbeit verwendeten Implementierung wurde der Zustand der Home-Komponente in globalen Variablen gespeichert. Die für diese notwendigen Zustandsänderungen wurden in Transitionen der Client-Komponenten modelliert. Dadurch ergibt sich ein symmetrisches System aus einer Menge von Client-Komponenten. Als Eigenschaft wurde in den Verifikationsexperimenten

mit diesem Testfall geprüft, dass bei exklusivem Cache-Zugriffsrecht einer Komponente auf eine Cachezeile, keine andere Komponente exklusives oder geteiltes Zugriffsrecht auf diese Cachezeile hat.

Als letzter Testfall wurde der in den Tabellen *MCL* genannte Testfall verwendet. Dieser ist eine Implementierung des in [145] präsentierten MCSLock-Ansatzes, zur Sicherstellung des wechselseitigen Ausschlusses für einen kritischen Abschnitt in einem asynchronen nebenläufigen System (siehe Abschnitt 2.3). Bei diesem Ansatz werden Sperrvariablen eingesetzt, um den Zutritt zum kritischen Abschnitt zu regeln. Ist gerade eine Komponente im kritischen Abschnitt, so wird durch gesetzte Sperrvariablen der Zutritt anderer Komponenten zum kritischen Abschnitt verhindert. Beim MCSLock-Ansatz werden auf den Eintritt zum kritischen Abschnitt wartende Komponenten in einer Liste verwaltet und es werden lokale Variablen der Komponenten als Sperrvariablen verwendet. Die eigenen lokalen Sperrvariablen werden von Komponenten, die auf Erlaubnis zum Eintritt in den kritischen Abschnitt warten, immer wieder abgefragt. Verlässt eine Komponente ihren kritischen Abschnitt, so setzt sie bei einer anderen wartenden Komponente die lokale Sperrvariable zurück und ermöglicht dadurch dieser den Zutritt zum kritischen Abschnitt. Durch die Benutzung lokaler Variablen von Komponenten als Sperrvariablen, wird der Kommunikationsaufwand des wiederholten Abfragens des Werts globaler Sperrvariablen durch Komponenten vermieden. Als Eigenschaft wurde in den Verifikationsexperimenten mit diesem Testfall der wechselseitige Ausschluss des Zutritts zum kritischen Abschnitt geprüft.

Bis auf den Testfall *MCL*, handelt es sich bei allen in diesem Abschnitt vorgestellten Testfällen um asynchrone nebenläufige Systeme mit Transitionslokalität (siehe Abschnitt 2.3). Beim Testfall *MCL* ist aufgrund des Rücksetzens der lokalen Sperrvariablen anderer Komponenten keine Transitionslokalität vorhanden. Da TLEBDDs und ETLEBDDs zur Repräsentation der Transitionsrelation von Systemen mit Transitionslokalität entworfen wurden, sind für den *MCL*-Testfall im Folgenden keine Ergebnisse von Verifikationsexperimenten mit TLEBDDs und ETLEBDDs aufgeführt. In allen hier beschriebenen Testfällen ist bis auf das Philosophenproblem vollständige Symmetrie vorhanden. Beim Philosophenproblem liegt Rotationssymmetrie vor (siehe Abschnitt 2.4.1.2 für eine Beschreibung der Symmetriearten).

8.2 Informationen zu den Verifikationsexperimenten

In den nachfolgenden Abschnitten werden Ergebnisse von Verifikationsexperimenten mit den im vorhergehenden Abschnitt beschriebenen Testfällen aufgeführt. Die Testfälle sind alle Implementierungen von asynchronen nebenläufigen Systemen aus replizierten Komponenten. In Tabellen mit Ergebnissen von Verifikationsexperimenten stehen die verwendeten Testfälle jeweils in der mit *Testfall* überschriebenen Spalte. Die bei einem Verifikationsexperiment verwendete Anzahl an replizierten Komponenten, ist ebenfalls in der Spalte *Testfall* hinter dem Namen eines verwendeten Testfalls angegeben. Bei der BDD-basierten Modellprüfung wird der bei einem Verifikationsexperiment maximal benötigte Speicherbedarf durch die maximale Anzahl während des Verifikationsexperiments gleich-

zeitig lebender und damit zu speichernder BDD-Knoten bestimmt. Diese Knotenanzahl muss zum erfolgreichen Abschluss eines Verifikationsexperiments gleichzeitig gespeichert werden können. Sie wird zu Verifikationsexperimenten in der mit *# BDD Knoten* beschrifteten Spalte angegeben. Ein bei einem Verifikationsexperiment aufgetretener Speicherüberlauf wird in den Tabellen durch den Eintrag von *Speicherül.* im entsprechenden Eintrag dieser Spalte angezeigt. Die für ein Verifikationsexperiment benötigte Laufzeit ist in den Tabellen in der mit *Zeit* beschrifteten Spalte zu finden. Bei Laufzeitangaben werden die Abkürzungen s, m und h für Sekunden, Minuten und Stunden verwendet. Falls ein Verifikationsexperiment nach 24 Stunden noch nicht beendet war, wurde es automatisch abgebrochen. In den Tabellen mit Ergebnissen von Verifikationsexperimenten wurde dafür für die Laufzeit solcher Verifikationsexperimente der Wert $> 24h$ eingetragen. In Tabellen zur Vorwärtserreichbarkeitsanalyse ist zusätzlich noch die Anzahl der für Verifikationsexperimente benötigten Bildberechnungen (*# Bildberechnungen*) aufgeführt. Diese entspricht der Anzahl der bei einem Verifikationsexperiment durchgeführten Berechnungen des relationalen Produkts. Zu Verifikationsläufen mit dynamischem Verändern der Variablenordnung ist außerdem die Anzahl der bei diesen durchgeführten Umordnungen der verwendeten Variablenordnung angegeben. Diese ist jeweils in der Spalte mit der Überschrift *# Umordnungen* zu finden. Weitere Informationen zu den Verifikationsexperimenten mit dynamischem Umordnen der Variablenordnung sind in Abschnitt 8.3 zu finden.

Die Verifikationsexperimente mit partitionierten Transitionsrelationen wurden mit komponentenweise partitionierten Transitionsrelationen durchgeführt. Dazu wurde in Verifikationsläufen mit ausschließlicher Verwendung von BDDs oder TLEBDDs, für jede in einem System vorhandene Komponente ein BDD bzw. TLEBDD mit allen Transitionen der Komponente angelegt. In einem ETLEBDD kann eine Menge isomorpher Transitionen gespeichert werden. Diese kann Transitionen mehrerer Komponenten beinhalten (siehe Kapitel 5). In den Verifikationsexperimenten mit ETLEBDDs wurde die Menge der im ETLEBDD gespeicherten isomorphen Transitionen maximal gewählt. Es wurden alle in einem System zwischen Komponenten vorhandenen isomorphen Transitionen mit dem ETLEBDD repräsentiert. Bei den für die Verifikationsexperimente mit ETLEBDDs verwendeten Testfällen waren vorhandene isomorphe Transitionen für alle Komponenten isomorph. Daher wurden sie in einem ETLEBDD gespeichert. Zur Repräsentation der übrigen nicht isomorphen Transitionen jeder Komponente, wurde jeweils für jede Komponente ein TLEBDD mit diesen Transitionen angelegt. In jedem in dieser Arbeit verwendeten Testfall sind isomorphe und nicht-isomorphe Transitionen vorhanden. Daher wurde in jedem Verifikationsexperiment mit der Benutzung eines ETLEBDDs für jede Komponente auch ein TLEBDD zur Repräsentation der nicht-isomorphen Transitionen verwendet. Zur Angabe der bei Verifikationsexperimenten verwendeten Transitionsrelationsarten wurden in den Tabellen mit den Ergebnissen von Verifikationsexperimenten entsprechende Abkürzungen benutzt. Diese sind in Tabelle 8.1 angegeben.

Bei der Vorwärtserreichbarkeitsanalyse werden wiederholt Berechnungen von Nachfolgerzuständen ausgeführt (siehe Abschnitt 2.1.6.3). In den Tabellen zur Vorwärtserreichbarkeitsanalyse ist für jeden Testfall neben der verwendeten Art der Repräsentation der

Tabelle 8.1: In den Tabellen mit Ergebnissen von Verifikationsexperimenten für Transitionsrelationsarten verwendete Abkürzungen.

Abkürzung	Verwendete Transitionsrelationsart
BDD Monolithisch	Repräsentation der gesamten Transitionsrelation durch einen monolithischen BDD.
BDD Partitioniert	Repräsentation der Transitionsrelation durch eine komponentenweise partitionierte Transitionsrelation aus BDDs. Für jede Komponente eines Systems wird ein BDD zur Repräsentation aller Transitionen der Komponente verwendet.
BDD Partitioniert ohne Identitätsmuster	Repräsentation der Transitionsrelation durch eine komponentenweise partitionierte Transitionsrelation aus BDDs. Für jede Komponente eines Systems wird ein BDD zur Repräsentation aller Transitionen der Komponente verwendet. In einem BDD für eine Komponente sind hier keine Knoten für Identitätsmuster zur Repräsentation von Änderungen des lokalen Zustands anderer Komponenten vorhanden.
TLEBDD Partitioniert	Repräsentation der Transitionsrelation durch eine komponentenweise partitionierte Transitionsrelation aus TLEBDDs. Für jede Komponente eines Systems wird ein TLEBDD zur Repräsentation aller Transitionen der Komponente verwendet.
ETLEBDD	Repräsentation der zwischen allen in einem System vorhandenen Komponenten isomorphen Transitionen in einem ETLEBDD. Für jede Komponente werden die restlichen Transitionen in einem TLEBDD gespeichert.

Transitionsrelation, noch der bei der Nachfolgerberechnung zur Berechnung des relationalen Produkts für die Nachfolgerberechnung verwendete Ansatz angegeben. Eine Zuordnung der dafür benutzten Abkürzungen zu den entsprechenden Ansätzen und eine Beschreibung dieser ist in Tabelle 8.2 zu finden. Bei den Verifikationsexperimenten mit Anwendung von Symmetriereduktionstechniken wurden auch Verifikationsexperimente mit zusätzlicher Ausnutzung von Zustandssymmetrien durchgeführt. In diesen Fällen ist dies in der Bezeichnung der entsprechenden Verifikationsexperimente nach der verwendeten Transitionsrelationsart und dem für Nachfolgerberechnungen verwendeten Ansatz mit angegeben. Als Beispiel würde ein Verifikationsexperiment zur Vorwärtserreichbarkeitsanalyse mit Symmetriereduktion, einer partitionierten Transitionsrelation aus BDDs, der Verwendung der kombinierten Berechnung des relationalen Produkts mit BDDs und zusätzlicher Ausnutzung von Zustandssymmetrien mit *BDD Partitioniert kombiniert mit Zustandssymmetrien* bezeichnet werden.

Tabelle 8.2: In den Tabellen mit Ergebnissen von Verifikationsexperimenten zur Vorwärtserreichbarkeitsanalyse für dabei verwendete Ansätze benutzte Abkürzungen.

Abkürzung	Für die Berechnung des relationalen Produkts verwendeter Ansatz
Standard	Separate aufeinanderfolgende Berechnung der im relationalen Produkt enthaltenen Operationen UND-Verknüpfung, existentielle Quantifizierung und Umbenennung von Knoten für Variablen zur Kodierung von Nachfolgerzuständen in Knoten für die zu diesen korrespondierenden Variablen für aktuelle Zustände.
kombiniert	Kombinierte Berechnung des relationalen Produkts. Für BDDs wird dazu Algorithmus 9 aus Abschnitt 3.1 und für TLEBDDs Algorithmus 12 aus Abschnitt 4.3 verwendet.
Teilresultate Bild, Teilresultate Alles	Kombinierte Berechnung des relationalen Produkts mit dem in dieser Arbeit vorgestellten neuen Verfahren zur Berücksichtigung bereits fertig berechneter Teilresultate. Für BDDs wird Algorithmus 23 aus Abschnitt 6.1 und für TLEBDDs Algorithmus 31 aus Abschnitt 10.1 verwendet. Als Explorationsstrategie wird beim Ansatz *Teilresultate Bild* die in Abschnitt 6.4 vorgestellte Strategie *BDD P. Part* (siehe Algorithmus 26) benutzt. Beim Ansatz *Teilresultate Alles* wird als Explorationsstrategie die in Abschnitt 6.4 vorgestellte Strategie *BDD P. Part All* (siehe Algorithmus 27) benutzt
Kombination Algorithmen	Berechnungen des relationalen Produkts mit den oben vorgestellten Verfahren *kombiniert*, *Teilresultate Bild* und *Teilresultate Alles*. Als Explorationsstrategie wird hier die in Abschnitt 6.4 vorgestellte Strategie *BDD P. ALG COMB* (siehe Algorithmus 28) zur dynamischen Kombination dieser Verfahren benutzt.
einfache simultane Exploration, mehrfache simultane Exploration	Verwendung von ETLEBDDs und TLEBDDs zur Repräsentation der Transitionsrelation. Für Nachfolgerberechnungen mit dem ETLEBDD wird Algorithmus 14 aus Abschnitt 5.3 zur simultanen Berechnung des relationalen Produkts für mehrere Komponenten verwendet. Die Nachfolgerberechnungen mit TLEBDDs werden mit Algorithmus 12 aus Abschnitt 4.3 durchgeführt. Als Explorationsstrategie wird beim Ansatz *einfache simultane Exploration* die in Abschnitt 5.8 vorgestellte und in Algorithmus 21 dargestellte einfache simultane Exploration benutzt. Beim Ansatz *mehrfache simultane Exploration* wird als Explorationsstrategie die in Abschnitt 5.8 vorgestellte und in Algorithmus 22 dargestellte mehrfache simultane Exploration benutzt.
Neu	Benutzung des neuen für partitionierte Transitionsrelationen in Algorithmus 29 in Abschnitt 7.1 vorgestellten Verfahrens, zur Erreichbarkeitsanalyse mit Benutzung der dynamischen symbolischen Symmetriereduktion.

8.3 Verwendete Variablenordnungen

In diesem Kapitel werden Ergebnisse von Verifikationsexperimenten zum ausschließlichen Bau der Transitionsrelation und zur Vorwärtserreichbarkeitsanalyse präsentiert. Dazu wurden sowohl Verifikationsexperimente mit einer vorab gewählten und fest verwendeten Variablenordnung, als auch Verifikationsexperimente mit dynamischem Verändern der Variablenordnung (siehe Abschnitt 2.2.2) durchgeführt. Bei den Verifikationsexperimenten mit fest gewählter Variablenordnung, wurde die in Abbildung 8.1 veranschaulichte Variablenordnung als verschachtelte Variablenordnung benutzt. Diese ist in der Abbildung für ein asynchrones nebenläufiges System aus m-Komponenten dargestellt. In verschachtelten Variablenordnungen (siehe Abschnitt 2.5.3) sind korrespondierende Variablen für aktuelle Zustände und Nachfolgerzustände immer direkt benachbart. Es ist weitläufig bekannt, dass solche Variablenordnungen bei der Modellprüfung häufig am effizientesten bezüglich benötigtem Speicherbedarf und Laufzeit sind. In der in Abbildung 8.1 veranschaulichten Variablenordnung kommen zuerst die Booleschen Variablen für aktuelle Zustände und Nachfolgerzustände b_{g1} bis $b_{g(2n_g)}$ zur Kodierung der Werte von globalen Variablen in Zuständen eines Systems vor. Anschließend sind nacheinander die Booleschen Variablen für aktuelle Zustände und Nachfolgerzustände zur Kodierung des lokalen Zustands der Komponenten angeordnet. Dabei kommen die Variablen zur Kodierung des lokalen Zustands einer Komponente in der Variablenordnung zusammenhängend vor. In Abbildung 8.1 steht b_{ij} bei Variablen zur Kodierung von lokalen Zuständen jeweils für die j-te Boolesche Variable zur Kodierung des lokalen Zustands von Komponente i.

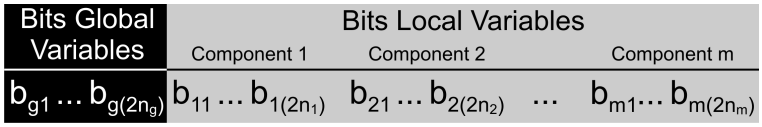

Abbildung 8.1: Veranschaulichung der bei den Verifikationsexperimenten ohne dynamisches Verändern der Variablenordnung fest verwendeten Variablenordnung.

In Verifikationsexperimenten mit dynamischem Verändern der Variablenordnung, wurde die in Abbildung 8.1 veranschaulichte Variablenordnung als Ausgangsvariablenordnung verwendet. Im vom Modellprüfer Sviss verwendeten BDD-Paket CUDD können Boolesche Variablen gruppiert werden. Bei Verwendung von Verfahren zum dynamischen Verändern der Variablenordnung mit Berücksichtigung der Gruppierungen, von denen ebenfalls welche in CUDD implementiert sind, müssen gruppierte Variablen dann in durch das dynamische Umordnen entstehenden Variablenordnungen weiterhin zusammenhängend vorhanden sein. Für alle Verifikationsexperimente mit dynamischem Verändern der Variablenordnung wurde in der vorliegenden Arbeit ein Sifting-Algorithmus mit Berücksichtigung und zusammenhängendem Verschieben von Gruppen von Variablen verwendet (siehe Abschnitt 2.2.2 und [155]). Bei Verifikationsexperimenten mit ausschließlicher Benutzung von BDDs, wurden die für das dynamische Umordnen erlaubten Variablenordnungen auf alle verschachtelten Variablenordnungen beschränkt. Die Beschränkung auf verschachtelte

Variablenordnungen wurde dabei durch Erzeugen einer Gruppe für jedes Variablenpaar aus korrespondierender Variable für aktuelle Zustände und Nachfolgerzustände erreicht. Dadurch werden durch das Verfahren zum dynamischen Umordnen aus [155] nur Variablenordnungen erzeugt, in denen die korrespondierenden Variablen benachbart sind und es werden nur verschachtelte Variablenordnungen erzeugt. Im BDD-Paket CUDD kann zusätzlich eingestellt werden, ob die Reihenfolge der Variablen innerhalb einer Gruppe in der Variablenordnung geändert werden darf. Das Vertauschen der Reihenfolge der gruppierten Variablen eines korrespondierenden Variablenpaares wurde bei den ausschließlich mit BDDs durchgeführten Verifikationsexperimenten zugelassen.

Für die Benutzung von TLEBDDs und ETLEBDDs wurden in den Abschnitten 4.2 (TLEBDDs) und 5.2 (ETLEBDDs) konforme Variablenordnungen definiert. Die in Abschnitt 4.3 für TLEBDDs und in Abschnitt 5.3 für ETLEBDDs jeweils für die kombinierte Berechnung des relationalen Produkts präsentierten Algorithmen, können mit nach diesen Definitionen konformen und zusätzlich verschachtelten Variablenordnungen verwendet werden. Für mit diesen Datenstrukturen durchgeführte Verifikationsexperimente mit dynamischem Umordnen der Variablenordnung, wurde ebenfalls der in [155] (siehe auch Abschnitt 2.2.2) vorgestellte Ansatz zum dynamischen Verändern der Variablenordnung mit Berücksichtigung von Gruppen von Variablen eingesetzt. Um mit diesem nur konforme und verschachtelte Variablenordnungen zu erhalten, wurde für die tatsächlichen Variablen eine Gruppe mit allen Variablen zur Kodierung des globalen Zustands und für jede Komponente eine Gruppe mit allen Variablen zur Kodierung des lokalen Zustands der Komponente erzeugt. Zusätzlich wurden Untergruppen für die Gruppe der globalen Variablen erzeugt. Für diese wurde eine Untergruppe für jedes Paar von korrespondierenden Variablen für aktuelle Zustände und Nachfolgerzustände angelegt. Innerhalb dieser Untergruppen wurde die Vertauschung der Reihenfolge der Variablen zugelassen. Außerdem wurde das Vertauschen der Reihenfolge dieser Untergruppen selbst erlaubt. Innerhalb der Gruppen mit Variablen zur Kodierung des lokalen Zustands von Komponenten wurden keine Vertauschungen von Variablen erlaubt. Der Grund dafür ist, dass zu reduzierten Variablen korrespondierende tatsächliche Variablen in der Variablenordnung beim Algorithmus für ETLEBDDs für alle Komponenten in der gleichen Reihenfolge vorkommen müssen. Dies würde bei Vertauschung der Reihenfolge der tatsächlichen Variablen zur Kodierung des lokalen Zustands für einzelne Komponenten verletzt werden. Vor Bildberechnungen mit TLEBDDs könnten dadurch erzeugte nicht für Bildberechnungen mit allen Komponenten gültigen Variablenordnungen durch erneutes Umordnen korrigiert werden. In dieser Arbeit wurde aber bei den Verifikationsexperimenten mit TLEBDDs ebenfalls die oben beschriebene auch für ETLEBDDs gültige Gruppierung von Variablen verwendet. Damit können erzwungene Umgruppierungen von Variablen und der dafür benötigte Zeitaufwand vermieden werden. Innerhalb der Gruppen aus Variablen zur Kodierung des lokalen Zustands von Komponenten wurden fix verschachtelte Variablenordnungen verwendet, wobei zuerst die Variablen für aktuelle Zustände bei korrespondierenden Variablenpaaren vorkommen. Die Variablenordnung der reduzierten Variablen und die Umordnungen für diese wurde für Variablen zur Kodierung des globalen Zustands an die Variablenordnung der entsprechenden tatsächlichen Variablen und die dort dafür verwendeten Gruppierungen gekoppelt.

Die Variablenordnung der reduzierten Variablen für lokale Zustände einer Komponente, wurde an die Variablenordnung der ersten Komponente für die entsprechenden tatsächlichen Variablen gekoppelt. Dadurch wurden die für die Konformität von verschachtelten Variablenordnungen für reduzierte Variablen und tatsächliche Variablen notwendigen Beziehungen sichergestellt. Mit den durch die Gruppierungen von Variablen für erlaubte Variablenordnungen eingeführten Restriktionen lassen sich nicht alle für die in dieser Arbeit für Bildberechnungen mit ETLEBDDs bzw. TLEBDDs präsentierten Algorithmen erlaubten Variablenordnungen erzeugen. Um die Menge der verwendbaren Variablenordnungen zu erweitern, könnte dazu ein neuer Algorithmus zum dynamischen Verändern der Variablenordnung bei Verwendung von TLEBDDs und ETLEBDDs entworfen werden.

8.4 Verifikationsexperimente zum Bau der Transitionsrelation

In den nachfolgenden Abschnitten werden experimentelle Ergebnisse zum Bau der Transitionsrelation präsentiert. Diese wurden mit den in Abschnitt 8.1 beschriebenen Testfällen für asynchrone nebenläufige Systeme aus replizierten Komponenten durchgeführt. Dazu wurden Verifikationsexperimente mit verschiedenen Datenstrukturen zur Repräsentation der Transitionsrelation und sowohl mit, als auch ohne dynamisches Verändern der Variablenordnung ausgeführt. Die zur Repräsentation der Transitionsrelation jeweils verwendete Datenstruktur ist in den im Folgenden aufgeführten Tabellen mit experimentellen Ergebnissen in der ersten Zeile angegeben (siehe auch Tabelle 8.1 für die für Transitionsrelationsarten dort verwendeten Abkürzungen). Ist bei einem Verifikationsexperiment ein Speicherüberlauf aufgetreten, wurde das in den Tabellen durch den Eintrag *Speicherül.* vermerkt. Die Laufzeit der hier ausgeführten Verifikationsexperimente wurde auf 24 Stunden beschränkt. Für Verifikationsexperimente, die nicht innerhalb von 24 Stunden beendet werden konnten, wurde in den Tabellen als Laufzeit $> 24h$ angegeben. Konnte ein Verifikationsexperiment mit einer bestimmten Komponentenanzahl aus einem der zuvor genannten Gründe nicht erfolgreich beendet werden, wurden für den verwendeten Testfall und die benutzte Art der Repräsentation der Transitionsrelation keine Verifikationsexperimente mit größerer Komponentenanzahl ausgeführt. In den in den Tabellen mit den experimentellen Ergebnissen dafür angegebenen Einträgen, wurde der Grund der nicht erfolgreichen Beendigung des letzten dafür ausgeführten Verifikationsexperiments eingetragen.

8.4.1 Bau Transitionsrelation ohne dynamisches Umordnen

In den Folgenden beiden Abschnitten werden experimentelle Ergebnisse zum Bau der Transitionsrelation ohne die Verwendung von Verfahren zum dynamischen Umordnen der Variablenordnung vorgestellt.

8.4.1.1 Bau Transitionsrelation mit BDDs

In Tabelle 8.3 sind Ergebnisse von Verifikationsexperimenten zum Bau von durch BDDs repräsentierten Transitionsrelationen angegeben. Die Ergebnisse der Verifikationsexperimente zeigen, dass für die Testfälle *MutL* und *MCL* eine durch einen monolithischen BDD repräsentierte Transitionsrelation für eine höhere Komponentenanzahl, als eine gewöhnliche partitionierte Transitionsrelation aus BDDs gebaut werden kann. Für beide Testfälle konnte mit der monolithischen Transitionsrelation dabei ein deutlich geringerer Speicherbedarf und für den *MCL*-Testfall ebenfalls ein große Reduktion der Laufzeit beobachtet werden. Beim *Mut*-Testfall konnte für die monolithische Transitionsrelation ein verringerter Speicherbedarf, bei allerdings deutlich größerer Laufzeit, beobachtet werden. Für alle anderen in der Tabelle angegebenen Testfälle kann die monolithische Transitionsrelation, unter den Ansätzen für die in Tabelle 8.3 experimentelle Ergebnisse dargestellt sind, für die geringsten Komponentenanzahlen gebaut werden. Außerdem führt deren Verwendung bei erfolgreich abgeschlossenen Experimenten zur höchsten maximal benötigten Anzahl an lebenden Knoten und auch zu den höchsten benötigten Laufzeiten.

Die für die oben genannten im Vergleich mit einer gewöhnlichen partitionierten Transitionsrelation besseren Ergebnisse der monolithischen Transitionsrelation, liegen an bei diesen Testfällen vorhandenen Vorteilen bei der Repräsentation von Identitätsmustern zur Repräsentation von Übergängen des lokalen Zustands von Komponenten bei einer monolithischen Transitionsrelation. In einer Partition einer komponentenweise partitionierten Transitionsrelation für eine Komponente, werden Identitätsmuster für Übergänge des lokalen Zustands anderer Komponenten, die vor den Teilen eines BDDs zur Repräsentation der Änderungen des lokalen Zustands der Komponente für die eine Partition gebaut wurde vorkommen, explizit repräsentiert. Die Übergänge der lokalen Zustände verschiedener Komponenten werden über unterschiedlichen Variablen kodiert. Aus diesem Grund liegen in der Regel für vor diesen vorkommende solche Identitätsmuster keine isomorphen Teilgraphen vor. Daher kann dort die übliche Fähigkeit von BDD-Programmpaketen isomorphe Teilgraphen nur einmal zu speichern noch nicht ausgenutzt werden. Bei einer monolithischen Transitionsrelation und den Testfällen *Mut*, *MutL* und *MCL* können Identitätsmuster oder Teile von Identitätsmustern häufig zur Kodierung von Identitätsübergängen des lokalen Zustands von Komponenten und zur Kodierung von Veränderungen des lokalen Zustands von Komponenten durch Transitionen gemeinsam verwendet werden. Daher können bei einer monolithischen Transitionsrelation hier Folgen solcher Identitätsmuster teilweise für mehrere Komponenten gemeinsam genutzt werden. In den Testfällen *Mut* und *MutL* sind viele solche Knoten für Identitätsmuster zur Kodierung von Änderungen des lokalen Zustands anderer Komponenten vorhanden. Dies ist bei den experimentellen Ergebnissen daran zu sehen, dass die bei diesen beiden Testfällen benötigte maximale Anzahl an lebenden Knoten, bei der Verwendung einer partitionierten Transitionsrelation aus BDDs ohne Identitätsmuster für Zustandsänderungen des lokalen Zustands anderer Komponenten deutlich geringer ist. Experimentelle Ergebnisse für diese Art der Repräsentation der Transitionsrelation sind in der dritten Spalte der Tabelle angegeben. Bei Repräsentation der Transitionsrelationen der Testfälle *Peterson*, *DP* und *CCP* führt der oben beschriebene Vorteil der Verwendung einer monolithischen Transitionsrelation zu

keiner Speicherreduktion. Es lässt sich bei einer monolithischen Transitionsrelation ein deutlich höherer Speicherbedarf als bei Verwendung einer gewöhnlichen partitionierten Transitionsrelation beobachten.

Tabelle 8.3: Bau Transitionsrelation ohne dynamisches Umordnen.

	BDD Monolithisch		BDD Partitioniert		BDD Partitioniert ohne Identitätsmuster	
Testfall	# BDD Knoten	Zeit	# BDD Knoten	Zeit	# BDD Knoten	Zeit
Mut 2665	57,052,625	2h 50m	85,614,006	1m 16s	403,205	0s
Mut 3800	Speicherül.	-	Speicherül.	-	506,708	0s
Mut 7500	Speicherül.	-	Speicherül.	-	1,120,698	1s
Mut 7850	Speicherül.	-	Speicherül.	-	Speicherül.	-
MutL 60	7,037,320	30s	37,610,014	21s	3,566,152	1s
MutL 110	39,597,374	4m 40s	268,333,592	3m 8s	14,966,756	8s
MutL 127	50,225,142	6m 29s	Speicherül.	-	17,276,560	9s
MutL 400	Speicherül.	-	Speicherül.	-	273,260,385	3m 17s
MutL 4000	Speicherül.	-	Speicherül.	-	Speicherül.	-
Pet 4	98,669,045	51s	332,478	0s	147,674	0s
Pet 8	Speicherül.	-	110,559,987	2m 12s	47,415,416	58s
Pet 9	Speicherül.	-	Speicherül.	-	115,675,212	2m 43s
DP 21	53,484,416	45s	13,498	0s	8,049	0s
DP 100	Speicherül.	-	278,652	0s	156,885	0s
DP 1000	Speicherül.	-	27,076,974	1h 33m	15,068,985	1h 20m
CCP 17	67,002,013	3m 33s	35,430,994	1m 50s	35,405,082	1m 48s
CCP 19	Speicherül.	-	157,337,844	10m 12s	157,305,291	9m 51s
CCP 22	Speicherül.	-	Speicherül.	-	Speicherül.	-
MCL 200	64,236	32m 42s	2,383,773	1h 43m	-	-
MCL 300	96,551	3h 8m	5,333,527	10h 31m	-	-
MCL 350	112,701	6h 3m	7,246,252	20h 25m	-	-
MCL 475	153,072	23h 50m	-	$>$ 24h	-	-

Die partitionierte Speicherung der Transitionsrelation ohne Repräsentation von Identitätstransformationen für Änderungen des lokalen Zustands von Komponenten, für die eine Partition der Transitionsrelation nicht gebaut wurde, liefert für alle Testfälle bis auf den Testfall *CCP* deutlich geringere maximal benötigte Knotenzahlen. Beim Testfall *CCP* ergibt sich für die gewöhnliche partitionierte Transitionsrelation mit BDDs eine ähnliche Knotenzahl. Geringe Unterschiede bei den für beide in Tabelle 8.3 aufgeführten Arten der Verwendung einer partitionierten Transitionsrelation benötigten Knotenanzahlen, treten bei wenigen Variablen zur Repräsentation des lokalen Zustands von Komponenten und wenigen verschiedenen Kofaktoren zur Repräsentation der Änderungen des lokalen Zustands von Komponenten auf. Sind wenige verschiedene Kofaktoren zur Repräsentation

der Zustandsänderungen des lokalen Zustands einer Komponente in einer Partition der Transitionsrelation vorhanden, so sind dazu nur die gleiche Anzahl an Folgen von Identitätsmustern für Übergänge lokaler Zustände anderer Komponenten, die zum Erreichen der Kodierung der Änderung des lokalen Zustands der Komponente für die die Partition der Transitionsrelation gebaut wurde führen, abzuspeichern. Eine Ausnahme bildet die Komponente, deren Änderungen der lokalen Zustände mit den als erstes in der verwendeten Variablenordnung zur Repräsentation des lokalen Zustands einer Komponente vorkommenden Variablen repräsentiert wird. Für diese kommen vor ihren Variablen zur Kodierung des lokalen Zustands keine Identitätsmuster für Änderungen des lokalen Zustands anderer Komponenten vor.

8.4.1.2 Bau Transitionsrelation mit TLEBDDs und ETLEBDDs

In Tabelle 8.4 sind Ergebnisse von Verifikationsexperimenten zum Bau der Transitionsrelation bei Benutzung von TLEBDDs (siehe Kapitel 4) und ETLEBDDs (siehe Kapitel 5) zur Repräsentation der Transitionsrelation angegeben. Die dort aufgeführten Ergebnisse zeigen, dass die Benutzung von TLEBDDs und ETLEBDDs gegenüber den für die Benutzung von BDDs in Tabelle 8.3 präsentierten Ergebnissen für alle Testfälle zu deutlich geringeren maximal benötigten Knotenanzahlen führt. Ebenso lassen sich für alle Testfälle Laufzeitverbesserungen beobachten. Mit Ausnahme des Philosophenproblems (DP), bei dem die Transitionsrelation auch mit den Ansätzen aus Abschnitt 8.4.1.1 für eine hohe Komponentenanzahl gebaut werden kann, kann die Transitionsrelation mit TLEBDDs und ETLEBDDs für alle Testfälle auch für eine höhere Komponentenanzahl gebaut werden.

Die großen Einsparungen beim Speicherbedarf bei der Benutzung von TLEBDDs und ETLEBDDs resultieren aus der Entfernung von Identitätsmustern und der Verwendung einer reduzierten Variablenmenge. Dadurch können größere isomorphe Teilgraphen entstehen und es ergeben sich zusätzliche Möglichkeiten der Ausnutzung des nur einmaligen Abspeicherns isomorpher Teilgraphen durch das BDD-Programmpaket. Das die zusätzliche Benutzung einer reduzierten Variablenmenge und die Umbenennung der tatsächlichen Variablen in diese Variablen bei TLEBDDs und ETLEBDDs zu einer großen Reduktion des Speicherbedarfs führt, sieht man beim Vergleich der experimentellen Ergebnisse mit den Ergebnissen bei der Benutzung von BDDs. Dort wurden experimentelle Ergebnisse für eine partitionierte Transitionsrelation, ohne die Speicherung von Identitätsmustern für Zustandsübergänge der lokalen Zustände von Komponenten, für die eine Partition der Transitionsrelation nicht gebaut wurde, präsentiert. Im Vergleich mit dieser lassen sich bei Benutzung von TLEBDDs und ETLEBDDs für alle Testfälle sehr große Reduktionen des Speicherbedarfs beobachten.

Bei den Verifikationsexperimenten mit Verwendung von ETLEBDDs, werden die vorhandenen nicht-isomorphen Transitionen durch einen TLEBDD für jede Komponente repräsentiert. Vergleicht man die Benutzung von ETLEBDDs und TLEBDDs mit der ausschließlichen Benutzung von TLEBDDs, ergibt sich durch das separate Abspeichern der isomorphen Transitionen in einem ETLEBDD für alle Testfälle sogar eine maximal benötigte Knotenanzahl, die leicht unter der bei der ausschließlichen Benutzung von TLEBDDs erzielten maximal benötigten Knotenanzahl liegt. Dies ist auf die separate Speicherung

der zwischen allen Komponenten isomorphen Transitionen im ETLEBDD zurückzuführen. Diese Transitionen müssen hier nur einmal gemeinsam im BDD des ETLEBDDs repräsentiert werden. Demgegenüber werden bei der ausschließlichen Repräsentation der Transitionsrelation durch TLEBDDs isomorphe und nicht-isomorphe Transitionen für jede Komponente gemeinsam in einem TLEBDD gespeichert. Durch die gemeinsame Speicherung von isomorphen und nicht-isomorphen Transitionen können isomorphe Teilgraphen möglicherweise nicht mehr in vollem Umfang ausgenutzt werden. Die Repräsentation der isomorphen Transitionen erfolgt nach Vermischung mit den vorhandenen nicht-isomorphen Transitionen möglicherweise nicht mehr durch zwischen den Komponenten vollständig isomorphe Teilgraphen.

Tabelle 8.4: Bau Transitionsrelation ohne dynamisches Umordnen.

	TLEBDD Partitioniert		ETLEBDD	
Testfall	# BDD Knoten	Zeit	# BDD Knoten	Zeit
Mut 2665	49,044	0s	45,708	0s
Mut 3800	67,806	0s	61,577	0s
Mut 7500	132,951	0s	121,593	0s
Mut 7850	139,805	0s	126,500	0s
MutL 60	65,420	0s	64,551	0s
MutL 110	146,141	0s	145,148	0s
MutL 127	145,421	0s	144,376	0s
MutL 400	719,237	14s	716,874	0s
MutL 4000	7,426,588	20m 49s	7,405,287	11s
Pet 4	93,280	0s	93,229	0s
Pet 8	17,403,225	42s	17,403,204	41s
Pet 9	40,089,105	1m 58s	40,087,456	1m 56s
DP 21	3,710	0s	3,454	0s
DP 100	42,766	0s	41,562	0s
DP 1000	3,118,888	1h 4m	3,106,884	1h 3m
CCP 17	4,924,302	58s	4,922,401	59s
CCP 19	19,671,729	4m 59s	19,669,445	4m 58s
CCP 22	157,300,375	1h 3m	157,297,449	1h 0m

8.4.2 Bau Transitionsrelation mit dynamischem Umordnen

In den Folgenden beiden Abschnitten werden experimentelle Ergebnisse zum Bau der Transitionsrelation mit Verwendung von Verfahren zum dynamischen Umordnen der Variablenordnung vorgestellt. Konnte die Transitionsrelation mit einer fest vorgegebenen Variablenordnung bei den für diese Arbeit dazu durchgeführten Verifikationsexperimenten nicht erfolgreich gebaut werden (siehe Abschnitt 8.4.1), war der Grund dafür zumeist

ein zu hoher für den Bau der Transitionsrelation benötigter Speicherbedarf. Wie an den zum Bau der Transitionsrelation mit Verwendung von Verfahren zum dynamischen Verändern der Variablenordnung nachfolgend präsentierten experimentellen Ergebnissen zu sehen ist, ist das Scheitern des Baus der Transitionsrelation hier dagegen oft auf eine zu hohe zum Bau der Transitionsrelation benötigte Laufzeit zurückzuführen (es wurde auch hier ein Zeitlimit von 24 Stunden verwendet).

8.4.2.1 Bau Transitionsrelation mit BDDs

Ergebnisse von Verifikationsexperimenten zum Bau der Transitionsrelation mit dynamischem Umordnen der Variablenordnung sind in Tabelle 8.5 zu finden. Dort sind experimentelle Ergebnisse für eine monolithische und eine partitionierte Transitionsrelation aus BDDs aufgeführt. Für eine komponentenweise partitionierte Transitionsrelation, bei der in Partitionen für eine Komponente keine Identitätsmuster zur Kodierung der Zustandsübergänge der lokalen Zustände anderer Komponenten abgespeichert werden, sind Ergebnisse von Verifikationsexperimenten in Tabelle 8.6 angegeben. Die deutlich geringsten Laufzeitwerte und die niedrigsten Werte für den benötigten Speicherbedarf lassen sich für die partitionierte Transitionsrelation bei Entfernung von Identitätsmustern beobachten. Für die Testfälle *MutL* und *CCP* lässt sich die Transitionsrelation mit diesem Ansatz auch für eine größere Komponentenanzahl, als es bei den anderen beiden Ansätzen möglich ist, bauen.

Vergleicht man die Verwendung einer monolithischen und einer partitionierten Transitionsrelation, so gibt es sowohl Testfälle, bei denen eine monolithische Transitionsrelation für eine höhere Komponentenanzahl erfolgreich gebaut werden kann (Testfall *Mut*), als auch Testfälle, bei denen eine partitionierte Transitionsrelation für eine höhere Komponentenanzahl erfolgreich gebaut werden kann (Testfall *DP*). Außerdem gibt es für beide Transitionsrelationsarten Testfälle, für die sie zur besseren Laufzeit bzw. zum geringeren Speicherbedarf führen. Die monolithische Transitionsrelation führt bei den Testfällen *Mut* und *MCL* zu deutlich besseren Ergebnissen, wohingegen bei der partitionierten Transitionsrelation erheblich bessere Ergebnisse für die Testfälle *MutL* und *DP* beobachtet werden können. Bei den Testfällen *Pet* und *CCP* hängt es von der Komponentenanzahl ab, welches Verfahren bessere Werte liefert. Das beide Verfahren für Testfälle zu den besten Ergebnissen führen können, kann hier darauf zurückzuführen sein, dass der performante Bau der Transitionsrelation sowohl durch die verwendete Transitionsrelationsart, als auch auf die Effizienz des Ansatzes zum Verändern der Variablenordnung, beeinflusst werden kann.

8.4.2.2 Bau Transitionsrelation mit TLEBDDs und ETLEBDDs

Ergebnisse von Verifikationsexperimenten zum Bau der Transitionsrelation mit TLEBDDs und ETLEBDDs mit dynamischem Verändern der Variablenordnung sind in Tabelle 8.6 zu finden. Die Ergebnisse zeigen, dass die Verwendung von ETLEBDDs dort für fast alle Verifikationsexperimente zu einem geringeren Speicherbedarf führt. Dieser ist meistens allerdings nur geringfügig unter dem mit TLEBDDs notwendigen Speicherbedarf. Die

Tabelle 8.5: Bau Transitionsrelation mit dynamischem Umordnen

	BDD Monolithisch			BDD Partitioniert		
Problem	# BDD Knoten	Zeit	# Umordnungen	# BDD Knoten	Zeit	# Umordnungen
Mut 3800	42,931,247	1h 21m	11	86,421,572	2h 21m	12
Mut 7500	35,155,424	1h 12m	10	Speicherül.	-	-
Mut 7850	Speicherül.	-	-	Speicherül.	-	-
MutL 60	10,740,312	15h 53m	39	7,466,473	6h 29m	35
MutL 600	-	> 24h	-	-	> 24h	-
MutL 4000	-	> 24h	-	-	> 24h	-
Pet 4	10,702	0s	9	6,517	0s	4
Pet 9	114,848	42s	24	110,268	39s	22
Pet 15	425,060	6m 15s	31	337,442	4m 23s	24
Pet 21	1,637,078	2h 20m	61	1,768,672	2h 34m	49
Pet 30	4,418,626	17h 5m	92	5,336,944	17h 53m	73
DP 21	12,176	0s	6	6,706	0s	5
DP 100	138,972	4m 25s	34	112,783	1m 38s	12
DP 1000	-	> 24h	-	26,057,659	2h 6m	11
CCP 19	42,496	9s	12	45,892	9s	10
CCP 22	44,481	10s	12	64,850	17s	13
CCP 100	2,420,586	4h 23m	73	1,747,227	11m 48s	27
CCP 500	-	> 24h	-	-	> 24h	-
CCP 950	-	> 24h	-	-	> 24h	-
MCL 200	655,302	36m 20s	39	2,730,860	1h 33m	39
MCL 300	1,160,066	2h 20m	26	2,474,255	4h 42m	25
MCL 475	-	> 24h	-	-	> 24h	-

Laufzeit ist mit ETLEBDDs ebenfalls bei nahezu allen Verifikationsexperimenten geringer. Für die Testfälle *CCP* und *Mut* lassen sich mit ETLEBDDs dabei hier auch größere Laufzeitverbesserungen beobachten.

Vergleicht man die in diesem Abschnitt für TLEBDDs und ETLEBDDs präsentierten Ergebnisse mit den für BDDs im letzten Abschnitt präsentierten experimentellen Ergebnissen, so liefern TLEBDDs und ETLEBDDs für alle Testfälle, bis auf den Testfall *Pet*, zumeist deutlich geringere maximal benötigte Knotenanzahlen und stark niedrigere Laufzeiten. Für den Testfall *Pet* lassen sich die besten Ergebnisse für die partitionierte Transitionsrelation bei Entfernung von Identitätsmustern erzielen.

Tabelle 8.6: Bau Transitionsrelation mit dynamischem Umordnen

	BDD Partitioniert ohne Identitätsmuster			TLEBDD			ETLEBDD		
Problem	# BDD Knoten	Zeit	# Umordnungen	# BDD Knoten	Zeit	# Umordnungen	# BDD Knoten	Zeit	# Umordnungen
Mut 3800	95,492	7s	2	61,715	2s	2	60,982	2s	2
Mut 7500	187,760	25s	3	187,760	35s	3	120,208	7s	2
Mut 7850	Speicherül.	-	-	-	> 24h	-	125,814	7s	2
MutL 60	973,942	6m 37s	20	65,092	7s	8	79,815	6s	8
MutL 600	12,180,734	17h 5m	21	1,318,176	6m 46s	10	1,658,064	5m 13s	10
MutL 4000	Speicherül.	-	-	-	> 24h	-	-	> 24h	-
Pet 4	6,517	0s	4	82,024	1s	6	81,331	1s	6
Pet 9	65,193	8s	13	35,677,675	4h 3m	16	35,115,147	4h 6m	16
Pet 15	199,134	48s	17	Speicherül.	-	-	-	> 24h	-
Pet 21	888,264	21m 58s	29	Speicherül.	-	-	-	> 24h	-
Pet 30	2,532,678	3h 50m	42	Speicherül.	-	-	-	> 24h	-
DP 21	4,446	0s	2	3,710	0s	0	3,454	0s	0
DP 100	64,242	19s	8	40,062	5s	4	40,601	6s	5
DP 1000	9,239,224	1h 27m	11	3,133,349	46m 18s	8	3,125,875	45m 43s	8
CCP 19	18,096	1s	7	10,226	0s	7	8,904	0s	5
CCP 22	26,664	2s	8	11,082	0s	8	11,142	0s	9
CCP 100	311,298	1m 35s	18	161,688	42s	19	157,683	37s	27
CCP 500	9,482,156	58m 49s	76	6,935,606	1h 54m	89	4,875,895	1h 17m	121
CCP 950	51,350,016	8h 22m	143	25,658,216	6h 47m	148	21,471,807	4h 8m	117

8.5 Vorwärtserreichbarkeitsanalyse ohne Symmetriereduktion

In den nachfolgenden Abschnitten werden Ergebnisse von Verifikationsexperimenten zur Vorwärtserreichbarkeitsanalyse ohne die Verwendung von Verfahren zur Symmetriereduktion präsentiert. Die dazu aufgeführten Tabellen mit experimentellen Ergebnissen enthalten in der ersten Zeile jeweils die verwendete Art der Transitionsrelation und das bei Bildberechnungen verwendete Verfahren. Erläuterungen zu den dazu verwendeten Begriffen sind in Abschnitt 8.2 zu finden. Nachfolgend werden in Abschnitt 8.5.1 zuerst Ergebnisse von Verifikationsexperimenten zur Vorwärtserreichbarkeitsanalyse mit einer fest gewählten Variablenordnung vorgestellt. Anschließend werden in Abschnitt 8.5.2 experimentelle Ergebnisse zur Vorwärtserreichbarkeitsanalyse mit dynamischem Verändern der Variablenordnung präsentiert.

8.5.1 Vorwärtserreichbarkeitsanalyse ohne dynamisches Umordnen

In den folgenden Abschnitten werden Ergebnisse von Verifikationsexperimenten zur Vorwärtserreichbarkeitsanalyse mit BDDs (Abschnitt 8.5.1.1), TLEBDDs (Abschnitt 8.5.1.2), ETLEBDDs (Abschnitt 8.5.1.3), dem neuen Verfahren zur Berücksichtigung von Teilresultaten mit BDDs (Abschnitt 8.5.1.4) und der Benutzung dieses Verfahrens mit TLEBDDs (Abschnitt 8.5.1.5) aufgeführt.

8.5.1.1 Vorwärtserreichbarkeitsanalyse mit BDDs

Hier werden experimentelle Ergebnisse zur Vorwärtserreichbarkeitsanalyse mit BDDs angegeben. Dazu enthält Tabelle 8.7 Ergebnisse von Verifikationsexperimenten bei Verwendung eines monolithischen BDDs zur Repräsentation der Transitionsrelation. In Tabelle 8.8 sind Ergebnisse von Verifikationsexperimenten, die mit einer partitionierten Transitionsrelation aus BDDs durchgeführt wurden, aufgeführt.

Die experimentellen Ergebnisse zeigen, dass die Verwendung der kombinierten Berechnung des relationalen Produkts gegenüber der sequentiellen Berechnung für alle verwendeten Testfälle und Verifikationsexperimente zu Laufzeitverbesserungen führt. Dies lässt sich sowohl für monolithische, als auch für partitionierte Transitionsrelationen aus BDDs beobachten. Bis auf den *Pet*-Testfall mit Verwendung einer monolithischen Transitionsrelation, lassen sich dabei sehr große Laufzeitverbesserungen erzielen. Für partitionierte Transitionsrelationen lassen sich für jeden Testfall sehr große Laufzeitverbesserungen, die auch immer größer als die Laufzeitverbesserungen bei den entsprechenden Verifikationsexperimenten mit einer monolithischen Transitionsrelation ausfallen, beobachten. Dies kann auf die größere auszuführende Anzahl an Bildberechnungen bei der Benutzung von partitionierten Transitionsrelationen zurückzuführen sein. Die dort bei Verifikationsexperimenten benötigte Anzahl an Bildberechnungen entspricht der Anzahl der für das entsprechende Verifikationsexperiment mit einer monolithischen Transitionsrelation benötigten Bildberechnungen, multipliziert mit der Anzahl an verwendeten Partitionen der Transitionsrelation. Für jede Bildberechnung kann die größere Effizienz der kombinierten Berechnung

des relationalen Produkts gegenüber der sequentiellen aufeinanderfolgenden Berechnung ausgenutzt werden.

Tabelle 8.7: Vorwärtserreichbarkeitsanalyse ohne dynamisches Umordnen

	BDD Monolithisch Standard			BDD Monolithisch kombiniert		
Testfall	# BDD Knoten	Zeit	# Bild-berech-nungen	# BDD Knoten	Zeit	# Bild-berech-nungen
Mut 70	1,007,346	3m 19s	75	337,061	23s	75
Mut 100	2,840,457	21m 44s	105	908,099	2m 28s	105
Mut 200	22,125,496	11h 56m	205	6,906,783	59m 8s	205
Mut 300	-	$>$ 24h	-	23,788,280	5h 48m	305
Mut 500	-	$>$ 24h	-	-	$>$ 24h	-
MutL 4	372,750	15s	98	166,995	3s	98
MutL 5	7,055,821	20m 27s	142	2,957,444	10m 49s	142
MutL 6	Speicherül.	-	-	60,441,077	9h 8m	195
MutL 7	Speicherül.	-	-	Speicherül.	-	-
Pet 4	98,669,142	1m 4s	47	98,669,142	1m 0s	47
Pet 5	Speicherül.	-	-	Speicherül.	-	-
DP 21	54,041,998	24m 15s	42	53,484,687	7m 46s	42
DP 23	Speicherül.	-	-	Speicherül.	-	-
CCP 7	6,124,916	7m 23s	58	1,390,036	2m 15s	58
CCP 8	22,885,419	39m 9s	66	4,786,048	14m 59s	66
CCP 9	83,789,250	3h 9m	74	16,551,428	1h 24m	74
CCP 11	Speicherül.	-	-	-	$>$ 24h	-
MCL 22	7,172,122	1h 6m	158	2,286,675	25m 54s	158
MCL 25	11,074,874	2h 28m	179	3,548,956	1h 6m	179
MCL 30	20,269,596	8h 1m	214	6,515,089	6h 38m	214
MCL 35	-	$>$ 24h	-	-	$>$ 24h	-

Zusätzlich führt die kombinierte Berechnung des relationalen Produkts zu einer Reduktion des maximal benötigten Speicherbedarfs. Bei Benutzung einer monolithischen Transitionsrelation kann der Speicherbedarf durch die kombinierte Berechnung des relationalen Produkts für jeden Testfall bis auf den *Pet*-Testfall reduziert werden. Für die monolithische Transitionsrelation ist die maximal benötigte Knotenanzahl beim *Pet*-Testfall für beide Arten der Berechnung des relationalen Produkts gleich. Dies ist bei diesem Testfall auf die zum Bau der monolithischen Transitionsrelation maximal benötigte Knotenanzahl zurückzuführen, die bei der Benutzung der kombinierten Berechnung des relationalen Produkts ebenfalls benötigt wird [1]. Für die partitionierte Transitionsrelation ergibt sich für

[1] Da bei den Verifikationsexperimenten zur Vorwärtserreichbarkeitsanalyse im Vergleich mit den Verifikationsexperimenten zum Bau der Transitionsrelation zusätzlich noch ein BDD mit Anfangszuständen

alle Verifikationsexperimente bis auf den *Pet*-Testfall für 4 Komponenten, bei dem die maximal benötigte Knotenanzahl wieder durch die zum Bau der Transitionsrelation benötigte Knotenanzahl bestimmt wird, eine Reduktion des Speicherbedarfs. Bei den Verifikationsexperimenten mit einer monolithischen Transitionsrelation werden durch die kombinierte Berechnung des relationalen Produkts größere Reduktionen des Speicherbedarfs erreicht. Dies lässt sich durch die bei Bildberechnungen mit monolithischen Transitionsrelationen in der Regel größeren auftretenden BDDs für Zwischenergebnisse, als die bei den entsprechenden Verifikationsexperimenten mit partitionierten Transitionsrelation auftretenden, erklären. Der Grund dafür ist, dass dort Bildberechnungen für alle Komponenten auf einmal ausgeführt werden, wodurch durch die einzelnen Bildberechnungen größere Zustandsmengen exploriert werden, die häufig in größere BDDs und höhere maximal benötigte Knotenanzahlen resultieren. Daher wirkt sich die Verwendung der kombinierten Berechnung des relationalen Produkts und die dadurch bei Bildberechnungen erzielbare Speicherersparnis bei den Verifikationsexperimenten mit der monolithischen Transitionsrelation stärker aus.

Im Folgenden werden die für monolithische und partitionierte Transitionsrelationen erzielten experimentellen Ergebnisse verglichen. Bei Verwendung einer partitionierten Transitionsrelation ergeben sich gegenüber einer monolithischen Transitionsrelation bis auf die Testfälle *MutL*, *Pet* und *CCP*, bei denen auch geringere Laufzeiten beobachtet wurden, größere Laufzeiten. Dabei lieferte die sequentielle aufeinanderfolgende Berechnung des relationalen Produkts mit partitionierten Transitionsrelationen nur für den *Pet*-Testfall eine bessere Laufzeit und ansonsten deutlich höhere Laufzeiten. Für die kombinierte Berechnung des relationalen Produkts lassen sich für die Testfälle *MutL*, *Pet* und *CCP* für partitionierte Transitionsrelationen niedrigere Laufzeiten beobachten. Die Laufzeitverbesserungen fallen dabei beim *Pet*-Testfall sehr groß aus. Bei Testfällen bei denen die partitionierte Transitionsrelation schlechtere Laufzeitwerte liefert, fallen diese nicht so hoch aus, wie bei der sequentiellen Berechnung des relationalen Produkts. Da bei partitionierten Transitionsrelationen eine größere Anzahl an Berechnungen des relationalen Produkts auszuführen sind, wirkt sich die Änderung der Art der Berechnung des relationalen Produkts hier stärker auf die Laufzeit aus. Der Speicherbedarf ist bei Verifikationsexperimenten mit der Benutzung einer partitionierten Transitionsrelation und der sequentiellen aufeinanderfolgenden Berechnung des relationalen Produkts immer geringer und oft deutlich niedriger als bei den entsprechenden Verifikationsexperimenten mit einer monolithischen Transitionsrelation. Bei der kombinierten Berechnung des relationalen Produkts lässt sich für einige Verifikationsexperimente ein deutlich geringerer Speicherbedarf mit der partitionierten Transitionsrelation beobachten (*Mut*, *Pet* und *DP*), wohingegen bei Verifikationsexperimenten mit anderen Testfällen der maximal benötigte Speicherbedarf bei Verwendung einer monolithischen Transitionsrelation geringer ist (*MutL*, *CCP* und *MCL*). Für die Testfälle *Pet* und *DP* führt die partitionierte Transitionsrelation mit der sequentiellen Berechnung des relationalen Produkts sogar zu einem geringeren Speicherbedarf, als die

und ein BDD mit Zuständen, die nicht erreicht werden sollen, angelegt werden, liegt die für die Vorwärtserreichbarkeitsanalyse beim *Pet*-Testfall angegebene maximale Knotenanzahl leicht über der maximalen Knotenanzahl beim Bau der Transitionsrelation.

kombinierte Berechnung bei der Benutzung einer monolithischen Transitionsrelation. Dies ist auf die große monolithische Transitionsrelation bei diesen Testfällen und die Reduktion des Speicherbedarfs für diese Transitionsrelationen bei Verwendung einer partitionierten Transitionsrelation zurückzuführen. Die Verwendung der kombinierten Berechnung des relationalen Produkts führt für diese Testfälle und die monolithischen Transitionsrelationen, aufgrund der Größe der monolithischen Transitionsrelationen, zu keiner signifikanten Reduktion des benötigten Speicherbedarfs.

Tabelle 8.8: Vorwärtserreichbarkeitsanalyse ohne dynamisches Umordnen

	BDD Partitioniert Standard			BDD Partitioniert kombiniert		
Testfall	# BDD Knoten	Zeit	# Bild-berech-nungen	# BDD Knoten	Zeit	# Bild-berech-nungen
Mut 70	745,322	3h 9m	5250	535,890	5m 8s	5250
Mut 100	-	> 24h	-	1,463,578	36m 12s	10500
Mut 200	-	> 24h	-	10,836,436	22h 46m	40400
Mut 300	-	> 24h	-	-	> 24h	-
MutL 4	273,802	32s	392	184,717	5s	392
MutL 5	4,716,311	28m 47s	710	3,012,077	9m 1s	710
MutL 6	-	> 24h	-	61,266,848	8h 45m	1170
MutL 7	-	> 24h	-	Speicherül.	-	-
Pet 4	332,576	4s	188	332,576	0s	188
Pet 5	1,481,270	4m 16s	370	1,287,785	29s	370
Pet 6	11,146,490	1h 51m	642	8,380,456	12m 42s	642
Pet 7	-	> 24h	-	101,714,419	7h 53m	1022
Pet 8	Speicherül.	-	-	Speicherül.	-	-
DP 21	5,891,614	2h 18m	882	3,706,116	12m 45s	882
DP 23	15,614,780	9h 29m	1058	9,803,484	44m 26s	1058
DP 28	-	> 24h	-	110,577,764	18h 24m	1596
DP 33	-	> 24h	-	Speicherül.	-	-
CCP 7	2,818,917	24m 17s	406	1,700,234	1m 58s	406
CCP 8	9,840,072	2h 28m	528	5,889,444	11m 9s	528
CCP 9	33,964,449	13h 43m	666	20,233,051	55m 47s	666
CCP 11	-	> 24h	-	228,267,388	22h 13m	990
CCP 15	-	> 24h	-	-	> 24h	-
MCL 22	6,480,412	15h 1m	3476	3,632,515	1h 17m	3476
MCL 25	-	> 24h	-	5,547,264	3h 23m	4475
MCL 30	-	> 24h	-	9,834,609	14h 14m	6420
MCL 35	-	> 24h	-	-	> 24h	-

8.5.1.2 Vorwärtserreichbarkeitsanalyse mit TLEBDDs

Für die vorliegende Arbeit wurden auch Verifikationsexperimente zur Vorwärtserreichbarkeitsanalyse bei Repräsentation der Transitionsrelation durch TLEBDDs durchgeführt. Dazu sind in Tabelle 8.9 Ergebnisse von Verifikationsexperimenten für die sequentielle und die kombinierte Berechnung des relationalen Produkts, bei Repräsentation der Transitionsrelation durch eine komponentenweise partitionierte Transitionsrelation aus TLEBDDs, angegeben.

Diese zeigen, dass die kombinierte Berechnung des relationalen Produkts für TLEBDDs für jeden Testfall zu sehr großen Laufzeitverbesserungen und zu einer Reduktion des maximal benötigten Speicherbedarfs führt. Für die Anwendung der entsprechenden beiden Verfahren zur Bildberechnung mit BDDs, konnte dies auch bei Verwendung einer partitionierten Transitionsrelation aus BDDs beobachtet werden (siehe Abschnitt 8.5.1.1). Beim Vergleich der in Abschnitt 8.5.1.1 für eine partitionierte Transitionsrelation aus BDDs präsentierten Ergebnisse von Verifikationsexperimenten mit den Ergebnissen für TLEBDDs, sind die Laufzeiten bei Verwendung von TLEBDDs und der sequentiellen Berechnung des relationalen Produkts zumeist ähnlich der für die entsprechenden Verifikationsexperimente mit BDDs benötigten Laufzeit. Eine deutlich geringere Laufzeit mit TLEBDDs lässt sich für den Testfall *MutL* für 6 Komponenten beobachten. Für die kombinierte Berechnung des relationalen Produkts und diesen beiden Arten der Repräsentation der Transitionsrelation, ergeben sich für alle Testfälle bei Verwendung von TLEBDDs geringere Laufzeiten und teilweise sehr große Laufzeitverbesserungen im Vergleich mit BDDs (bis zu 45 Prozent). Die geringeren Laufzeiten mit TLEBDDs resultieren vermutlich aus den in TLEBDDs nicht repräsentierten Identitätsmustern. Diese müssen in Berechnungen des relationalen Produkts mit TLEBDDs zum Beispiel nicht mit durchlaufen werden, wodurch der dafür notwendige Zeitbedarf eingespart werden kann. Mit einer partitionierten Transitionsrelation aus TLEBDDs lässt sich im Vergleich mit der partitionierten Transitionsrelation aus BDDs, ebenfalls für jedes Verifikationsexperiment und für beide Arten der Berechnung des relationalen Produkts eine Verringerung des maximal benötigten Speicherbedarfs erreichen. Besonders große Reduktionen des Speicherbedarfs lassen sich für den *MutL*- und den *Peterson*-Testfall beobachten. Ein Grund dafür ist die Speicherersparnis bei Repräsentation von Transitionsrelationen mit TLEBDDs. Neben einer Reduktion des maximal benötigten Speicherbedarfs kann es mit TLEBDDs auch möglich werden, TLEBDDs für größeren Teilmengen der Transitionsrelation zu erzeugen, als es bei der Verwendung von BDDs möglich wäre. Dies kann zu zusätzlichen Laufzeitverbesserungen bei der Berechnung des relationalen Produkts mit den größeren Mengen von Transitionen führen.

Die Berechnung des relationalen Produkts mit TLEBDDs führt nie zu schlechteren Laufzeiten, als die entsprechende Art der Berechnung des relationalen Produkts mit partitionierten Transitionsrelationen aus BDDs. Daher schneiden TLEBDDs beim Vergleich der Laufzeiten mit den entsprechenden für eine monolithische Transitionsrelation aus BDDs erzielten Laufzeiten besser ab, als die partitionierte Transitionsrelation aus BDDs. Für die kombinierte Berechnung des relationalen Produkts, lässt sich beispielsweise für TLEBDDs im Vergleich zur monolithischen Transitionsrelation für alle erfolgreich beendeten Verifikationsexperimente, bis auf die mit dem *Mut*-Testfall, dem *MutL*-Testfall für 4 Komponen-

ten und die mit dem *DP*-Testfall für 21 Komponenten, eine bessere Laufzeit erzielen. Die Laufzeitverbesserungen fallen dabei sehr groß aus. Durch die Reduktion des Speicherbedarfs bei der Benutzung von TLEBDDs gegenüber der Verwendung einer partitionierten Transitionsrelation aus BDDs, kann mit TLEBDDs bei der kombinierten Berechnung des relationalen Produkts zum Beispiel beim Testfall *MutL* ein geringerer maximaler Speicherbedarf (der für 6 Komponenten sehr groß ausfällt) als bei der monolithischen Transitionsrelation erzielt werden. Demgegenüber liefert die monolithische Transitionsrelation hier einen geringeren Speicherbedarf als die partitionierte Transitionsrelation aus BDDs.

Tabelle 8.9: Vorwärtserreichbarkeitsanalyse ohne dynamisches Umordnen

	TLEBDD Partitioniert Standard			TLEBDD Partitioniert kombiniert		
Testfall	# BDD Knoten	Zeit	# Bild-berech-nungen	# BDD Knoten	Zeit	# Bild-berech-nungen
Mut 70	568,902	3h 9m	5250	472,745	3m 42s	5250
Mut 100	-	> 24h	-	1,337,914	25m 59s	10500
Mut 200	-	> 24h	-	10,343,454	16h 26m	40400
Mut 300	-	> 24h	-	-	> 24h	-
MutL 4	221,648	30s	392	162,007	4s	392
MutL 5	3,802,945	26m 37s	710	2,978,359	5m 40s	710
MutL 6	53,409,206	17h 12m	1170	43,295,446	3h 27m	1170
MutL 7	Speicherül.	-	-	Speicherül.	-	-
Pet 4	101,916	3s	188	93,339	0s	188
Pet 5	731,762	3m 9s	370	583,965	25s	370
Pet 6	8,660,266	1h 51m	642	6,431,019	12m 8s	642
Pet 7	-	> 24h	-	89,401,785	5h 54m	1022
Pet 8	-	> 24h	-	Speicherül.	-	-
DP 21	5,786,244	2h 20m	882	3,696,331	10m 2s	882
DP 23	15,402,833	9h 9m	1058	9,791,671	33m 9s	1058
DP 28	-	> 24h	-	110,560,441	14h 6m	1596
DP 33	-	> 24h	-	Speicherül.	-	-
CCP 7	2,555,891	24m 19s	406	1,683,035	1m 33s	406
CCP 8	9,118,473	2h 27m	528	5,855,163	8m 40s	528
CCP 9	31,942,540	13h 21m	666	20,162,240	42m 38s	666
CCP 11	-	> 24h	-	227,946,256	16h 39m	990
CCP 15	-	> 24h	-	-	> 24h	-

8.5.1.3 Vorwärtserreichbarkeitsanalyse mit ETLEBDDs

Zur Erreichbarkeitsanalyse mit Verwendung von ETLEBDDs zur Repräsentation der zwischen den Komponenten vorhandenen isomorphen Transitionen, wurden in Abschnitt 5.8

die Strategien *einfache simultane Exploration* und *mehrfache simultane Exploration* vorgestellt. Für beide Strategien sind in Tabelle 8.10 Ergebnisse von Verifikationsexperimenten zu finden. In den Tabellen ist sowohl die Anzahl der insgesamt mit ETLEBDDs und den zusätzlich vorhandenen TLEBDDs (siehe Abschnitt 8.2) durchgeführten Bildberechnungen (abgekürzt mit *Gesamt*), als auch die Anzahl der ausschließlich mit ETLEBDDs ausgeführten Bildberechnungen (abgekürzt mit *ETLE.*) angegeben.

Tabelle 8.10: Vorwärtserreichbarkeitsanalyse ohne dynamisches Umordnen

	ETLEBDD einfache simultane Exploration				ETLEBDD mehrfache simultane Exploration			
Testfall	# BDD Knoten	Zeit	# Bildberechnungen		# BDD Knoten	Zeit	# Bildberechnungen	
			Gesamt	ETLE.			Gesamt	ETLE.
Mut 70	315,745	19s	5325	75	22,674	0s	285	75
Mut 100	865,513	2m 6s	10605	105	44,619	0s	405	105
Mut 200	6,741,556	47m 31s	41205	205	171,168	1s	805	205
Mut 300	23,420,671	4h 50h	91805	305	380,524	8s	1205	305
Mut 500	-	> 24h	-	-	1,020,104	18s	2005	505
Mut 1000	-	> 24h	-	-	4,045,949	2m 31s	4505	1005
Mut 1500	-	> 24h	-	-	9,314,226	8m 13s	5005	1505
MutL 4	159,063	3s	490	98	216,454	2s	181	149
MutL 5	2,916,910	9m 16s	852	142	3,406,806	2m 45s	328	268
MutL 6	59,303,294	7h 50m	1365	195	64,272,453	1h 56m	552	450
MutL 7	Speicherül.	-	-	-	Speicherül.	-	-	-
Pet 4	93,327	0s	235	47	93,327	0s	207	47
Pet 5	576,889	26s	444	74	520,864	19s	399	74
Pet 6	6,310,736	12m 52s	749	107	5,546,787	9m 33s	683	107
Pet 7	88,107,278	6h 44m	1168	146	76,201,888	4h 59m	1077	146
Pet 8	Speicherül.	-	-	-	Speicherül.	-	-	-
DP 21	3,156,467	6m 10s	924	42	501,082	7s	273	42
DP 23	8,369,859	21m 12s	1104	46	1,300,510	28s	322	46
DP 28	94,644,598	7h 42m	1653	57	13,329,035	8m 57s	477	57
DP 33	-	> 24h	-	46	144,141,323	2h 38m	627	66
CCP 7	1,480,622	1m 4s	464	58	469,185	7s	288	78
CCP 8	5,367,797	5m 51s	594	66	1,573,417	40s	361	89
CCP 9	19,185,060	27m 20s	740	74	5,385,365	3m 21s	442	100
CCP 11	Speicherül.	-	-	-	59,807,864	1h 11m	628	122
CCP 15	Speicherül.	-	-	-	Speicherül.	-	-	-

Die in Tabelle 8.10 angegebenen Ergebnisse von Verifikationsexperimenten zeigen, dass die Benutzung der *mehrfachen simultanen Exploration* gegenüber der *einfachen simultanen Exploration*, bei allen Verifikationsexperimenten zu sehr großen Laufzeitverbesserun-

gen führt. Ebenso wird durch diese Strategie bei den Testfällen *Mut*, *DP* und *CCP* eine sehr große Reduktion der maximal benötigten Anzahl an lebenden BDD-Knoten erreicht. Ein Anstieg der maximal benötigten Knotenanzahl ist dagegen nur beim *MutL*-Testfall zu beobachten, bei dem die Knotenanzahl mit der *mehrfachen simultanen Exploration* leicht höher ist. Für alle Verifikationsexperimente ergibt sich außerdem bei der *mehrfachen simultanen Exploration* eine geringere Anzahl an insgesamt durchgeführten Bildberechnungen. Dies resultiert aus den bei dieser Strategie wiederholt mit dem ETLEBDD aufeinanderfolgend ausgeführten Bildberechnungen. Durch Bildberechnungen mit dem ETLEBDD werden hier gleichzeitig für alle Komponenten, für die die im ETLEBDD gespeicherten Transitionen gültig sind, Bildberechnungen ausgeführt. Dadurch können durch diese Bildberechnungen große Mengen an neu gefundenen Zuständen entdeckt werden, die zum einen bei der *mehrfachen simultanen Exploration* durch nachfolgende Bildberechnungen mit dem ETLEBDD direkt weiter exploriert werden. Zum anderen können sich durch die Bildberechnungen mit den ETLEBDDs große Zustandsmengen ansammeln, die anschließend durch Bildberechnungen mit den TLEBDDs direkt weiterexploriert werden. Dadurch können auch bei diesen Bildberechnungen größere Zustandsmengen weiterexploriert werden.

Im Vergleich mit den in den vorangehenden Abschnitten (Abschnitt 8.5.1.1 und Abschnitt 8.5.1.2) für die Vorwärtserreichbarkeitsanalyse vorgestellten Ergebnissen von Verifikationsexperimenten, liefert die Strategie der *mehrfachen simultanen Exploration* für alle Verifikationsexperimente die geringste Laufzeit. Außerdem ergibt sich mit dieser Strategie für alle Testfälle, bis auf den Testfall *MutL*, ebenfalls der geringste maximal benötigte Speicherbedarf. Die einfache simultane Exploration besitzt im Vergleich mit den experimentellen Ergebnissen bei einer Repräsentation der Transitionsrelation durch einen monolithischen BDD, für alle Verifikationsexperimente bis auf die kombinierte Berechnung des relationalen Produkts für den Testfall *MutL* mit 4 und 6 Komponenten eine bessere Laufzeit. Der Speicherbedarf ist mit der *einfachen simultanen Exploration* für alle Verifikationsexperimente, bis auf die kombinierte Berechnung des relationalen Produkts und den Testfall *CCP*, ebenfalls geringer. Beim Vergleich der *einfachen simultanen Exploration* mit den Verifikationsexperimenten für partitionierte Transitionsrelationen aus BDDs und TLEBDDs, liefert diese Strategie für alle Testfälle bis auf die Testfälle *MutL* und *Pet* geringere Laufzeiten. Für die Testfälle *MutL* und *Pet* ergeben sich bei manchen Verifikationsexperimenten mit den partitionierten Transitionsrelationen aus BDDs und TLEBDDs bessere Laufzeiten. Ein geringerer maximal benötigter Speicherbedarf lässt sich mit partitionierten Transitionsrelationen nur für den Testfall *MutL* für 6 Komponenten bei Verwendung von TLEBDDs beobachten.

Die guten Laufzeitwerte bei der Verwendung von ETLEBDDs sind auf den in Abschnitt 5.3 vorgestellten Algorithmus zur simultanen Bildberechnung für mehrere Komponenten und das dabei mögliche Ausnutzen von für mehrere Komponenten gleichen Teilberechnungen zurückzuführen. Mit diesem Algorithmus können außerdem bei Bildberechnungen mit ETLEBDDs größere Zustandsmengen exploriert werden. Dies wird bei der Explorationsstrategie *mehrfache simultane Exploration* durch erschöpfendes Ausführen von Bildberechnungen mit dem ETLEBDD und das dadurch erfolgende Ansammeln einer großen

mit den TLEBDDs weiter zu explorierenden Zustandsmenge ausgenutzt. Wie die experimentellen Ergebnisse zeigen, führt dies zu weiteren sehr großen Laufzeitverbesserungen.

8.5.1.4 Vorwärtserreichbarkeitsanalyse neues Verfahren Teilresultate mit BDDs

Hier werden Ergebnisse von Verifikationsexperimenten mit dem in Kapitel 6 eingeführten neuen Verfahren, zur Berücksichtigung bereits fertig berechneter Teilresultate während Bildberechnungen, präsentiert. Für die Benutzung des dazu für BDDs in Abschnitt 6.1 vorgestellten Algorithmus bei der Vorwärtserreichbarkeitsanalyse, wurden in Abschnitt 6.4 drei verschiedene Strategien angegeben. Erklärungen zu den in den dafür angegebenen Tabellen mit experimentellen Ergebnissen verwendeten Abkürzungen für Verfahren, sind in Abschnitt 8.2 zu finden. Von den drei in Abschnitt 6.4 vorgestellten Strategien, können zwei Strategien bei der Verwendung einer monolithischen Transitionsrelation benutzt werden. Ergebnisse von Verifikationsexperimenten mit diesen sind in Tabelle 8.11 angegeben. Bei Verwendung partitionierter Transitionsrelationen können hingegen alle drei in Abschnitt 6.4 vorgestellten Strategien benutzt werden. Dafür sind in Tabelle 8.12 und in Tabelle 8.13 experimentelle Ergebnisse aufgeführt.

Mit dem Algorithmus mit ausschließlicher Benutzung der neuen Berücksichtigung von Teilresultaten für eine monolithische Transitionsrelation (*BDD Monolithisch Teilresultate Bild*), lässt sich im Vergleich zur kombinierten Berechnung des relationalen Produkts mit einer monolithischen Transitionsrelation (siehe Tabelle 8.7) nur für den Testfall *CCP* für alle Verifikationsexperimente eine bessere Laufzeit beobachten. Zusätzlich führt dieses Verfahren beim Testfall *MCL* mit 30 Komponenten zu einer sehr großen Laufzeitverbesserung, wohingegen bei diesem Testfall für 22 und 25 Komponenten eine Erhöhung der Laufzeit zu beobachten ist. Bei diesem Testfall scheinen sich die durch die bei Bildberechnungen beteiligten BDDs gegebenen Eigenschaften während der Vorwärtserreichbarkeitsanalyse in Abhängigkeit der Komponentenanzahl zu verändern. Für den *MutL*-Testfall mit 4 Komponenten resultiert die gleiche Laufzeit und bei allen weiteren Verifikationsexperimenten liefert die kombinierte Berechnung des relationalen Produkts bessere Laufzeiten. Der Ansatz mit der dynamischen Auswahl des für Bildberechnungen zu verwendenden Verfahrens (*BDD Monolithisch Kombination Algorithmen*) führt für alle Testfälle, bis auf die Testfälle *DP* und *MCL*, zu ähnlichen Laufzeiten wie der beste der beiden kombinierten Ansätze bei dessen ausschließlicher Benutzung. Für die Testfälle *DP* und *MCL* können gegenüber dem besten Verfahren bei ausschließlicher Verwendung sogar Laufzeitverbesserungen beobachtet werden. Diese fallen für das Verifikationsexperiment mit dem *MCL*-Testfall für 35 Komponenten sehr groß aus. Bezüglich des Speicherbedarfs liefern beide Varianten zur Verwendung des neuen Algorithmus zur Berücksichtigung bereits fertig berechneter Teilresultate eine ähnliche maximale Anzahl an benötigten BDD-Knoten, wie sie bei der kombinierten Berechnung des relationalen Produkts mit einer monolithischen Transitionsrelation aus einem BDD auftreten.

Bei Verwendung einer partitionierten Transitionsrelation ergibt für beide in Abschnitt 6.4 dafür vorgestellten Strategien zur ausschließlichen Benutzung des neuen Verfahrens zur Berücksichtigung von Teilresultaten, wie bei der Benutzung einer monolithischen Transitionsrelation, mit der neuen Strategie für die Testfälle *Mut* und *MutL* eine Laufzeiterhöhung

Tabelle 8.11: Vorwärtserreichbarkeitsanalyse ohne dynamisches Umordnen

	BDD Monolithisch Teilresultate Bild			BDD Monolithisch Kombination Algorithmen		
Testfall	# BDD Knoten	Zeit	# Bild-berech-nungen	# BDD Knoten	Zeit	# Bild-berech-nungen
Mut 70	320,539	45s	75	337,061	27s	75
Mut 100	893,211	5m 1s	105	908,099	2m 36s	105
Mut 200	6,782,502	2h 53m	205	6,906,783	1h 1m	205
Mut 300	-	> 24h	-	23,788,280	5h 55m	305
Mut 500	-	> 24h	-	-	> 24h	-
MutL 4	157,415	3s	98	166,979	3s	98
MutL 5	2,716,873	13m 2s	142	2,957,426	11m 35s	142
MutL 6	54,851,961	12h 50m	195	60,441,057	9h 38m	195
MutL 7	Speicherül.	-	-	Speicherül.	-	-
Pet 4	98,669,142	1m 1s	47	98,669,142	1m 0s	47
Pet 5	Speicherül.	-	-	Speicherül.	-	-
DP 21	53,484,687	8m 56s	42	53,484,687	5m 49s	42
DP 23	Speicherül.	-	-	Speicherül.	-	-
CCP 7	1,380,161	2m 5s	58	1,380,536	2m 9s	58
CCP 8	4,786,048	12m 44s	66	4,786,048	13m 26s	66
CCP 9	16,551,428	1h 8m	74	16,551,428	1h 17m	74
CCP 11	-	> 24h	-	-	> 24h	-
MCL 22	2,050,743	33m 39s	158	2,050,743	34m 56s	158
MCL 25	3,139,779	1h 16m	179	3,139,779	1h 12m	179
MCL 30	5,758,663	2h 59m	214	5,758,663	2h 29m	214
MCL 35	-	> 24h	-	9,744,881	4h 55m	249

im Vergleich zur kombinierten Berechnung des relationalen Produkts mit einer partitionierten Transitionsrelation aus BDDs. Zusätzlich erhöht sich hier die Laufzeit noch für den *Pet*-Testfall. Für die Testfälle *Mut* und *MutL* weist die Strategie *BDD Partitioniert Teilresultate Bild* dabei deutlich geringere Laufzeiten als die Strategie *BDD Partitioniert Teilresultate Alles* auf. Laufzeitverbesserungen werden mit beiden Varianten der ausschließlichen Benutzung des neuen Algorithmus zur Berücksichtigung von Teilresultaten für die Testfälle *DP* und *CCP* erzielt. Sehr große Laufzeitverringerungen, die auch höher als bei der Strategie *BDD Partitioniert Teilresultate Bild* ausfallen, lassen sich für die Testfälle *DP* und *CCP* beim Verfahren *BDD Partitioniert Teilresultate Alles* beobachten. Beim Testfall *MCL* konnten geringere Laufzeiten gegenüber der kombinierten Berechnung des relationalen Produkts für alle Verifikationsexperimente nur für die Strategie *BDD Partitioniert Teilresultate Bild* erzielt werden. Mit der Strategie *BDD Partitioniert Teilresultate Alles* ergibt sich nur für das Verifikationsexperiment mit diesem Testfall für 22 Komponenten eine Laufzeitverbesserung, sonst sind Laufzeiterhöhungen zu beobachten. Bei der

Benutzung des Verfahrens *BDD Partitioniert Teilresultate Alles* ergeben sich außerdem für die Verifikationsexperimente Verringerungen der maximal benötigten Anzahl an lebenden BDD-Knoten. Größere Verringerungen des maximal benötigten Speicherbedarfs treten dabei bei den Testfällen *DP* und *CCP* auf. Dies kann auf die hier nicht notwendige explizite Veroderung, der bereits bei Bildberechnungen mit anderen Partitionen der Transitionsrelation neu gefundenen Zustände, mit dem Resultat einer Bildberechnung zurückzuführen sein. Dadurch muss die dazu notwendige *ODER*-Operation, bei der die maximal benötigte Anzahl an BDD-Knoten beeinflussende große Zwischenergebnisse auftreten können, nicht ausgeführt werden. Die bei den Verifikationsexperimenten mit der Strategie *BDD Partitioniert Teilresultate Bild* ermittelten maximal benötigten Knotenanzahlen sind bis auf den Testfall *MCL*, bei dem sich für alle Verifikationsexperimente eine Verringerung der maximal notwendigen Knotenanzahl ergibt, der Anzahl der maximal benötigten Knoten beim Algorithmus zur kombinierten Berechnung des relationalen Produkts ähnlich.

Das Verfahren *BDD Partitioniert Kombination Algorithmen* versucht dynamisch den für Bildberechnungen aktuell effizientesten Algorithmus auszuwählen. Die Benutzung dieses Ansatzes führt bei nahezu allen Verifikationsexperimenten zu Laufzeiten, die ähnlich der Laufzeit des in ihm kombinierten Verfahrens mit der geringsten Laufzeit bei ausschließlicher Benutzung sind. Beim Testfall *MCL* ergeben sich gegenüber der ausschließlichen Anwendung der Einzelverfahren damit sogar sehr große Laufzeitverbesserungen. Vergleicht man die Anwendung des neuen Verfahrens zur Berücksichtigung von Teilresultaten für partitionierte und monolithische Transitionsrelationen, dann sieht man, dass die Laufzeitverbesserungen für partitionierte Transitionsrelationen deutlich höher ausfallen. Andererseits ergeben sich bei Verifikationsexperimenten mit Laufzeiterhöhungen aber auch größere Laufzeiterhöhungen, als bei der monolithischen Transitionsrelation. Dies kann auf die höhere Anzahl an durchzuführenden Bildberechnungen bei der Benutzung partitionierter Transitionsrelationen zurückzuführen sein. Dadurch wirken sich Laufzeitverbesserungen und Laufzeitverschlechterungen, die bei einzelnen Bildberechnungen anfallen, insgesamt möglicherweise stärker aus.

Die Laufzeitzuwächse bei Verwendung des neuen Verfahrens der Berücksichtigung von Teilresultaten bei Verifikationsexperimenten, könnten auf die für dessen Anwendung notwendige Veränderung bei der Benutzung der Ergebnistabelle zurückzuführen sein. Um die Eindeutigkeit von Zugriffen auf die Ergebnistabelle sicherzustellen ist es hier zusätzlich nötig, den in einem Rekursionsschritt übergebenen BDD für ein bereits fertig berechnetes Teilresultat zu berücksichtigen (siehe Abschnitt 6.1). Dies kann die Trefferanzahl bei Zugriffen auf die Ergebnistabelle verringern. Für einen Treffer muss hier beim Zugriff auf die Ergebnistabelle noch der BDD mit dem vorhandenen Teilresultat übereinstimmen. Bei einer niedrigeren Trefferquote bei Zugriffen auf die Ergebnistabelle, müssen statt der Rückgabe eines gespeicherten Ergebnisses die dazu notwendigen Berechnungen durchgeführt werden. Damit fällt auch die dafür notwendige Laufzeit an. Die experimentellen Ergebnisse zeigen auch, dass je nach Testfall ein anderes Verfahren der Benutzung des Algorithmus mit dem neuen Verfahren zur Berücksichtigung von Teilresultaten vorteilhaft sein kann, oder sogar deren Kombination (siehe Testfall *MCL*). Daher kann die Entwicklung von weiteren effizienten Verfahren zur dynamischen Kombination der verschiedenen

Tabelle 8.12: Vorwärtserreichbarkeitsanalyse ohne dynamisches Umordnen

	BDD Partitioniert Teilresultate Bild			BDD Partitioniert Teilresultate Alles		
Testfall	# BDD Knoten	Zeit	# Bild-berech-nungen	# BDD Knoten	Zeit	# Bild-berech-nungen
Mut 70	535,890	24m 1s	5250	515,056	41m 34s	5250
Mut 100	1,463,578	3h 29m	10500	1,289,465	6h 14m	10500
Mut 200	-	> 24h	-	-	> 24h	-
MutL 4	184,701	6s	392	175,516	6s	392
MutL 5	3,012,059	9m 10s	710	2,778,239	10m 50s	710
MutL 6	61,266,828	9h 29m	1170	56,543,512	18h 3m	1170
MutL 7	Speicherül.	-	-	-	> 24h	-
Pet 4	332,576	0s	188	332,576	0s	188
Pet 5	1,287,785	34s	370	1,277,705	31s	370
Pet 6	8,380,456	14m 44s	642	8,101,647	13m 36s	642
Pet 7	101,714,419	9h 18m	1022	94,715,793	8h 52m	1022
Pet 8	Speicherül.	-	-	Speicherül.	-	-
DP 21	3,706,116	10m 23s	882	3,008,783	6m 28s	882
DP 23	9,803,484	35m 39s	1058	7,954,722	23m 46s	1058
DP 28	110,577,764	14h 51m	1596	89,688,585	8h 15m	15967
DP 33	Speicherül.	-	-	Speicherül.	-	-
CCP 7	1,700,234	1m 48s	406	1,428,542	1m 35s	406
CCP 8	5,889,444	9m 29s	528	4,921,121	7m 47s	528
CCP 9	20,233,051	47m 51s	666	16,851,950	38m 28s	666
CCP 11	228,267,388	18h 35m	990	189,939,928	13h 55m	990
CCP 15	-	> 24h	-	-	> 24h	-
MCL 22	2,887,899	1h 4m	3476	3,004,177	1h 2m	3476
MCL 25	4,399,943	2h 46m	4475	4,588,360	4h 23m	4475
MCL 30	7,856,491	11h 45m	6420	8,195,103	15h 58m	6420
MCL 35	-	> 24h	-	-	> 24h	-

Algorithmen, die versuchen den aktuell besten Algorithmus dynamisch auszuwählen, zu weiteren Laufzeitsteigerungen führen. Ebenso könnte auch der Wechsel der Strategie zur Berücksichtigung von Teilresultaten innerhalb eines Algorithmus zur Bildberechnung zu weiteren Laufzeitverbesserungen führen.

8.5.1.5 Vorwärtserreichbarkeitsanalyse neues Verfahren Teilresultate mit TLEBDDs

In diesem Abschnitt werden experimentelle Ergebnisse zu Verifikationsexperimenten mit dem in Algorithmus 31 in Abschnitt 10.1 für TLEBDDs vorgestellten Algorithmus zur kombinierten Berechnung des relationalen Produkts, mit dem neuen Verfahren zur Be-

Tabelle 8.13: Vorwärtserreichbarkeitsanalyse ohne dynamisches Umordnen

	BDD Partitioniert Kombination Algorithmen		
Testfall	# BDD Knoten	Zeit	# Bild-berech-nungen
Mut 70	535,890	5m 19s	5250
Mut 100	1,463,578	36m 26s	10500
Mut 200	10,836,436	23h 32m	41000
Mut 300	-	> 24h	-
MutL 4	183,296	6s	392
MutL 5	3,012,059	9m 10s	710
MutL 6	61,266,828	9h 5m	1170
MutL 7	Speicherül.	-	-
Pet 4	332,576	0s	188
Pet 5	1,287,785	31s	370
Pet 6	8,380,456	13m 11s	642
Pet 7	101,714,419	8h 23m	1022
Pet 8	Speicherül.	-	-
DP 21	3,410,698	6m 34s	882
DP 23	9,041,428	22m 28s	1058
DP 28	101,819,125	8h 33m	1596
DP 33	Speicherül.	-	-
CCP 7	1,700,234	1m 32s	406
CCP 8	5,514,596	7m 44s	528
CCP 9	19,020,049	36m 14s	666
CCP 11	228,267,388	13h 38m	990
CCP 15	-	> 24h	-
MCL 22	3,004,177	41m 5s	3476
MCL 25	5,314,560	1h 48m	4475
MCL 30	8,483,061	6h 9m	6420
MCL 35	-	> 24h	-

rücksichtigung bereits fertig berechneter Teilresultate, präsentiert. Dazu werden Ergebnisse von Verifikationsexperimenten mit den drei in Abschnitt 6.4 präsentierten Ansätzen zur Verwendung dieses Verfahrens angegeben. In Tabelle 8.14 sind experimentelle Ergebnisse mit den beiden in Abschnitt 6.4 vorgestellten Arten der ausschließlichen Benutzung des Algorithmus mit dem neuen Verfahren zur Berücksichtigung von Teilresultaten aufgeführt. Ergebnisse von Verifikationsexperimenten mit dem ebenfalls in diesem Abschnitt vorgestellten Ansatz mit der gemeinsamen Verwendung dieser beiden Varianten und dem Algorithmus zur kombinierten Berechnung des relationalen Produkts mit TLEBDDs sind

in Tabelle 8.15 aufgeführt. Weitere Erklärungen zu diesen Ansätzen und zu den für diese in den Tabellen verwendeten Abkürzungen sind in Abschnitt 8.2 zu finden.

Tabelle 8.14: Vorwärtserreichbarkeitsanalyse ohne dynamisches Umordnen

	TLEBDD Teilresultate Bild			TLEBDD Teilresultate Alles		
Testfall	# BDD Knoten	Zeit	# Bild-berech-nungen	# BDD Knoten	Zeit	# Bild-berech-nungen
Mut 70	472,745	14m 57s	5250	451,911	25m 24s	5250
Mut 100	1,337,914	2h 11m	10500	1,163,801	3h 49m	10500
Mut 200	-	> 24h	-	-	> 24h	-
MutL 4	162,007	4s	392	152,822	4s	392
MutL 5	2,978,359	5m 3s	710	2,744,530	5m 29s	710
MutL 6	61,481,397	4h 5m	1170	60,638,781	6h 1m	1170
MutL 7	Speicherül.	-	-	Speicherül.	-	-
Pet 4	93,339	0s	188	93,339	0s	188
Pet 5	583,965	25s	370	573,885	24s	370
Pet 6	6,431,019	12m 6s	642	6,152,210	11m 28s	642
Pet 7	90,660,489	6h 13m	1022	84,720,021	5h 37m	1022
Pet 8	Speicherül.	-	-	Speicherül.	-	-
DP 21	3,696,331	8m 16s	882	2,998,998	5m 2s	882
DP 23	9,791,671	27m 55s	1058	7,942,909	17m 35s	1058
DP 28	110,560,041	10h 53m	1596	89,670,862	6h 20m	1596
DP 33	Speicherül.	-	-	Speicherül.	-	-
CCP 7	1,683,035	1m 17s	406	1,411,343	1m 8s	406
CCP 8	5,855,163	7m 41s	528	4,886,840	6m 0s	528
CCP 9	20,162,240	35m 44s	666	16,781,139	28m 28s	666
CCP 11	227,946,256	13h 57m	990	189,618,796	10h 40m	990
CCP 15	-	> 24h	-	-	> 24h	-

Die in Tabelle 8.14 angegebenen Ergebnisse von Verifikationsexperimenten zeigen, dass der Algorithmus mit dem neuen Verfahren der Berücksichtigung von Teilresultaten nur für den Testfall *Mut* für beide Varianten der ausschließlichen Benutzung eine schlechtere Laufzeit als die kombinierte Berechnung des relationalen Produkts mit TLEBDDs (siehe Tabelle 8.9) liefert. Die Laufzeit ist bei diesem Testfall für beide Strategien der Verwendung des neuen Verfahrens zur Berücksichtigung von Teilresultaten deutlich schlechter. Für den Testfall *MutL* liefern beide neuen Ansätze für 4 und 5 Komponenten die gleiche Laufzeit bzw. nahezu die gleiche Laufzeit und für 6 Komponenten eine schlechtere Laufzeit als die kombinierte Berechnung des relationalen Produkts mit einem TLEBDD. Bei den restlichen Testfällen *Pet*, *DP* und *CCP*, liefert die Verwendung der beiden Varianten der neuen Art der Benutzung von Teilresultaten hingegen jeweils bessere Laufzeiten. Eine Ausnahme ist lediglich das Verifikationsexperiment mit dem *Pet*-Testfall für 7 Kompo-

Tabelle 8.15: Vorwärtserreichbarkeitsanalyse ohne dynamisches Umordnen

	TLEBDD Kombination Algorithmen		
Testfall	# BDD Knoten	Zeit	# Bild-berech-nungen
Mut 70	472,745	3m 45s	5250
Mut 100	1,337,914	26m 30s	10500
Mut 200	10,343,454	17h 18m	41000
Mut 300	-	> 24h	-
MutL 4	162,007	5s	392
MutL 5	2,978,359	5m 27s	710
MutL 6	61,805,329	4h 23m	1170
MutL 7	Speicherül.	-	-
Pet 4	93,339	0s	188
Pet 5	580,752	25s	370
Pet 6	6,427,097	12m 6s	642
Pet 7	89,231,612	5h 43m	1022
Pet 8	Speicherül.	-	-
DP 21	3,393,381	4m 52s	882
DP 23	9,011,684	16m 47s	1058
DP 28	101,774,131	6h 10m	1596
DP 33	Speicherül.	-	-
CCP 7	1,678,705	1m 10s	406
CCP 8	5,627,809	6m 1s	528
CCP 9	19,576,012	27m 56s	666
CCP 11	226,889,508	11h 6m	990
CCP 15	-	> 24h	-

nenten, für das die Strategie *TLEBDD Teilresultate Bild* zu einer schlechteren Laufzeit als die gewöhnliche kombinierte Berechnung des relationalen Produkts führt. Sehr große Laufzeitverbesserungen ergeben sich dabei mit beiden Varianten der Benutzung des neuen Algorithmus bei den Testfällen *DP* und *CCP*.

Beim Vergleich der Verfahren *TLEBDD Teilresultate Bild* und *TLEBDD Teilresultate Alles* untereinander zeigt sich, dass die Laufzeitverschlechterungen gegenüber der kombinierten Berechnung des relationalen Produkts mit TLEBDDs bei den Testfällen *Mut* und *MutL* mit dem Verfahren *TLEBDD Teilresultate Bild* geringer als beim Verfahren *TLEBDD Teilresultate Alles* ausfallen. Ebenso fallen aber auch die bei den anderen Verifikationsexperimenten beobachteten Laufzeitverbesserungen bei diesem geringer als beim Verfahren *TLEBDD Teilresultate Alles* aus. Zusätzlich können bei der Strategie *TLEBDD Teilresultate Alles*, bis auf den Testfall *MutL*, unter allen Strategien für die in diesem Abschnitt experimentelle Ergebnisse präsentiert werden und der gewöhnlichen kombinier-

ten Berechnung mit TLEBDDs die geringsten maximal benötigten Anzahlen lebender Knoten beobachtet werden. Bei der Strategie *TLEBDD Teilresultate Bild* ist hingegen, ebenfalls mit Ausnahme des Testfalls *MutL*, fast immer die gleiche maximale Anzahl lebender Knoten wie bei der gewöhnlichen kombinierten Berechnung des relationalen Produkts aufgetreten. Beim *MutL*-Testfall weist die gewöhnliche kombinierte Berechnung für 6 Komponenten eine deutlich geringere Knotenanzahl als die anderen Ansätze auf.

Das Verfahren, das die beiden Verfahren der Berücksichtigung von Teilresultaten und die kombinierte Berechnung des relationalen Produkts mit einem TLEBDD gemeinsam verwendet, liefert für nahezu jedes Verifikationsexperiment eine ähnliche oder eine etwas über dem besten Verfahren bei ausschließlicher Benutzung liegende Laufzeit. Die besten Laufzeiten aller hier betrachteten Ansätze lassen sich damit für den *DP*-Testfall und den *CCP*-Testfall mit 9 Komponenten erzielen. Beim Vergleich der Varianten der ausschließlichen Benutzung des Algorithmus mit dem neuen Verfahren der Berücksichtigung von Teilresultaten zwischen TLEBDDs und BDDs liefert deren Benutzung für TLEBDDs für alle Testfälle bessere Laufzeiten. Dies wurde bereits beim Vergleich der in dieser Arbeit für TLEBDDs und BDDs vorgestellten Algorithmen zur kombinierten Berechnung des relationalen Produkts beobachtet. Ebenso kann mit diesen Verfahren für TLEBDDs bei den meisten Verifikationsexperimenten ein geringerer maximal benötigter Speicherbedarf als mit BDDs beobachtet werden.

8.5.2 Vorwärtserreichbarkeitsanalyse mit dynamischem Umordnen

In den Folgenden Abschnitten werden Ergebnisse zur Vorwärtserreichbarkeitsanalyse mit Verwendung von Verfahren zum dynamischen Umordnen der Variablenordnung präsentiert. Als Verfahren zum dynamischen Verändern der Variablenordnung wurde ein Sifting-Algorithmus mit Berücksichtigung und zusammenhängendem Verschieben von Gruppen von Variablen benutzt (siehe Abschnitt 2.2.2 und [155]). Als für diesen Ansatz erlaubte Variablenordnungen wurden die in Abschnitt 8.3 dazu beschriebenen Variablenordnungen verwendet. In den nachfolgenden Abschnitten werden Ergebnisse von Verifikationsexperimenten mit BDDs (Abschnitt 8.5.2.1), TLEBDDs (Abschnitt 8.5.2.2), und ETLEBDDs (Abschnitt 8.5.2.3) vorgestellt.

8.5.2.1 Vorwärtserreichbarkeitsanalyse mit BDDs

In Tabelle 8.16 und in Tabelle 8.17 sind Ergebnisse von Verifikationsexperimenten zur Vorwärtserreichbarkeitsanalyse unter Verwendung der kombinierten Berechnung des relationalen Produkts mit dynamischem Verändern der Variablenordnung angegeben. Dafür sind in den Tabellen experimentelle Ergebnisse mit Repräsentationen der Transitionsrelation durch einen monolithischen BDD (*BDD Monolithisch kombiniert*) und durch eine partitionierte Transitionsrelation aus BDDs (*BDD Partitioniert kombiniert*) aufgeführt.

An den experimentellen Ergebnissen ist zu sehen, dass die Verifikationsexperimente hier nie aufgrund eines zu großen Speicherbedarfs nicht erfolgreich beendet werden konnten. Wenn das erfolgreiche Durchführen eines Verifikationsexperiments nicht möglich war, lag dies immer an einer über der verwendeten oberen Laufzeitgrenze von 24 Stunden liegenden benötigten Laufzeit. Dies ist vermutlich auf das dynamische Umordnen der Variablenordnung zurückzuführen, bei dem das Ermitteln möglichst günstiger Variablenordnungen einen hohen Zeitbedarf in Anspruch nehmen kann. Die Vorwärtserreichbarkeitsanalyse kann mit einer partitionierten Transitionsrelation bei allen Verifikationsexperimenten, für mindestens die gleiche oder eine höhere Komponentenanzahl, als für eine monolithische Transitionsrelation erfolgreich durchgeführt werden. Der Grund dafür sind bei Verifikationsexperimenten mit höheren Komponentenanzahlen zumeist deutlich geringere Laufzeiten mit der partitionierten Transitionsrelation. Die monolithische Transitionsrelation liefert bei den Verifikationsexperimenten, die mit ihr erfolgreich beendet werden können, für die höheren Komponentenanzahlen oft einen geringeren maximal benötigten Speicherbedarf. Allerdings ist der maximal benötigte Speicherbedarf bei keinem der in Tabelle 8.16 und in Tabelle 8.17 aufgeführten Verifikationsexperimente kritisch. Dieser ist weit vom Speicherbedarf, ab dem Verifikationsexperimente auf dem verwendeten Rechner nicht mehr erfolgreich beendet werden können, entfernt.

Im Vergleich mit den entsprechenden Verifikationsexperimenten ohne dynamisches Verändern der Variablenordnung, liegt der maximal benötigte Speicherbedarf sowohl für die monolithische, als auch für die partitionierte Transitionsrelation, mit dynamischem Umordnen sehr stark unter dem bei den gleichen Verifikationsexperimenten dort benötigten maximalen Speicherbedarf (siehe Tabelle 8.7 für die monolithische Transitionsrelation und Tabelle 8.8 für die partitionierte Transitionsrelation). Durch das Ermöglichen der Ver-

wendung kleinerer BDDs, kann auch die Bildberechnung mit BDDs schneller durchgeführt werden. Zusätzlich ist aber bei der Benutzung von Verfahren zum dynamischen Umordnen der Variablenordnung, noch der Zeitaufwand zum Ermitteln möglichst günstiger Variablenordnungen zu berücksichtigen. Mit dynamischem Umordnen der Variablenordnung lassen sich für die monolithische und die partitionierte Transitionsrelation für alle Verifikationsexperimente der Testfälle *Pet*, *DP* und *CCP* sehr große Laufzeitverbesserungen gegenüber einer fest verwendeten Variablenordnung erzielen. Für die partitionierte Transitionsrelation lassen sich im Vergleich zu den gleichen Verifikationsexperimenten ohne dynamisches Umordnen, auch noch für den Testfall *Mut* sehr große Laufzeitverbesserungen erzielen. Eine deutliche Erhöhung der Laufzeit mit dynamischem Umordnen lässt sich hingegen für beide Arten der Transitionsrelation für den Testfall *MCL* beobachten. Beim Testfall *Mut* für die monolithische Transitionsrelation und dem Testfall *MutL* für beide Arten der Transitionsrelation, zeigt sich durch die Verwendung des dynamischen Umordnens der Variablenordnung bei den niedrigeren angegebenen Komponentenanzahlen eine Laufzeitverbesserung. Für die höheren angegebenen Komponentenanzahlen ergibt sich mit dynamischem Umordnen eine deutliche Laufzeitverschlechterung. Dies ist möglicherweise auf die dort größere Anzahl an Variablen über denen die BDDs definiert sind und den größeren Aufwand zum Finden günstiger Variablenordnungen zurückzuführen.

Bei allen Verifikationsexperimenten mit dynamischem Umordnen, auch bei denen bei welchen mit dynamischem Umordnen für bestimmte Komponentenanzahlen deutlich bessere Laufzeitwerte erzielt werden können, zeigt sich, dass die Laufzeit bei höheren Komponentenanzahlen sehr stark ansteigen kann. Dadurch konnten hier Verifikationsexperimente nicht mehr im zur Verfügung gestellten Zeitrahmen ($24h$) erfolgreich beendet werden. Bei einer fest gewählten Variablenordnung erwies sich hingegen oft der benötigte Speicherbedarf als kritisch. Dadurch kann es auch sein, dass mit dynamischem Umordnen bei einem Verifikationsexperiment für bestimmte Komponentenanzahlen deutlich geringere Laufzeiten als ohne dynamisches Umordnen erreicht werden können und das aber Verifikationsexperimente für höhere Komponentenanzahlen nur noch ohne dynamisches Umordnen erfolgreich beendet werden können. Dies ist der Fall, wenn der Speicherbedarf ohne dynamisches Umordnen noch nicht zu hoch ist und konnte hier zum Beispiel für den *MutL*-Testfall beobachtet werden.

8.5.2.2 Vorwärtserreichbarkeitsanalyse mit TLEBDDs

Für die Vorwärtserreichbarkeitsanalyse mit TLEBDDs mit dynamischem Umordnen der Variablenordnung sind in Tabelle 8.18 Ergebnisse von Verifikationsexperimenten aufgeführt. Mit dynamischem Umordnen ergibt sich für die kombinierte Berechnung des relationalen Produkts, im Vergleich mit den mit TLEBDDs ohne dynamisches Umordnen durchgeführten entsprechenden Verifikationsexperimenten, für alle Verifikationsexperimente bis auf die mit dem Testfall *DP* eine Laufzeiterhöhung. Die Laufzeiterhöhungen fallen dabei oft sehr hoch aus und für den Testfall *DP* liefert die Verwendung des dynamischen Umordnens eine Reduktion der Laufzeit. Der maximal benötigte Speicherbedarf ist mit dynamischem Umordnen, bis auf die Verifikationsexperimente mit dem *MutL*-Testfall für 4 Komponenten, mit dem *Pet*-Testfall mit 5 Komponenten und mit dem *CCP*-Testfall

Tabelle 8.16: Vorwärtserreichbarkeitsanalyse mit dynamischem Umordnen

	BDD Monolithisch kombiniert				BDD Partitioniert kombiniert			
Problem	# BDD Knoten	Zeit	# Bild-berech-nungen	# Umord-nungen	# BDD Knoten	Zeit	# Bild-berech-nungen	# Umord-nungen
Mut 100	248,974	22s	105	11	554,030	15m 17s	10500	12
Mut 200	-	>24h	-	-	1,983,696	3h 48m	41000	33
Mut 300	-	>24h	-	-	-	>24h	-	-
MutL 4	126,996	10s	98	10	133,667	10s	392	9
MutL 5	1,562,166	8m 56s	142	14	1,632,375	3m 31s	710	14
MutL 6	-	>24h	-	-	-	> 24h	-	-
Pet 4	18,033	0s	47	10	16,419	0s	188	7
Pet 5	205,020	9s	74	15	107,089	7s	370	12
Pet 6	625,415	1m 30s	107	21	740,885	2m 15s	642	17
Pet 7	3,564,374	36m 25s	146	23	5,549,877	34m 15s	1022	17
Pet 8	-	> 24h	-	-	-	> 24h	-	-
DP 20	36,669	2s	41	9	36,608	2s	820	6
DP 21	47,793	3s	42	9	39,789	2s	882	6
DP 23	66,201	4s	46	11	53,188	5s	1058	8
DP 25	47,591	3s	50	9	62,676	5s	1250	7
DP 28	97,112	7s	57	12	88,913	12s	1596	9
DP 30	122,411	10s	61	13	93,916	14s	1830	8
DP 50	379,706	9m 52s	101	20	297,621	3m 37s	5050	14
DP 100	1,351,757	3h 38m	201	40	1,442,150	42m 13s	20100	15
DP 120	-	> 24h	-	-	1,745,884	1h 25m	28920	16
DP 130	-	> 24h	-	-	-	> 24h	-	-

Tabelle 8.17: Vorwärtserreichbarkeitsanalyse mit dynamischem Umordnen

	BDD Monolithisch kombiniert				BDD Partitioniert kombiniert			
Problem	# BDD Knoten	Zeit	# Bild-berech-nungen	# Umord-nungen	# BDD Knoten	Zeit	# Bild-berech-nungen	# Umord-nungen
CCP 8	139,781	13s	66	12	281,742	25s	528	12
CCP 9	338,932	2m 41s	74	14	233,934	26s	666	11
CCP 11	854,939	33m 21s	90	16	775,095	2m 34s	990	14
CCP 15	-	$>$ 24h	-	-	27,324,909	14h 50m	122	1830
CCP 17	-	$>$ 24h	-	-	-	$>$ 24h	-	-
MCL 22	1,908,883	10h 5m	158	14	2,548,638	7h 27m	3476	12
MCL 25	-	$>$ 24h	-	-	3,892,774	15h 50m	4,475	16
MCL 27	-	$>$ 24h	-	-	-	$>$ 24h	-	-

Tabelle 8.18: Vorwärtserreichbarkeitsanalyse mit dynamischem Umordnen

	TLEBDD Partitioniert kombiniert			
Testfall	# BDD Knoten	Zeit	# Bild-berech-nungen	# Umord-nungen
Mut 100	1,180,381	33m 21s	10500	23
Mut 200	-	> 24h	-	-
MutL 4	167,167	8s	392	8
MutL 5	2,673,467	13m 20s	710	18
MutL 6	-	> 24h	-	-
Pet 4	88,482	1s	188	6
Pet 5	587,930	48s	370	9
Pet 6	6,288,278	29m 5s	642	12
Pet 7	-	> 24h	-	-
DP 20	2,174,219	6m 34s	820	11
DP 21	3,559,615	17m 33s	882	11
DP 23	10,231,059	2h 28m	1058	13
DP 25	26,397,727	16h 24m	1250	15
DP 28	-	-	> 24h	-
CCP 8	6,136,502	4h 33m	528	18
CCP 9	-	> 24h	-	-

für 8 Komponenten, geringer. Für die genannten Verifikationsexperimente mit den Testfällen *MutL*, *Pet* und *CCP* ist eine leichte Erhöhung des Speicherbedarfs im Vergleich mit den entsprechenden Verifikationsexperimenten ohne dynamisches Umordnen zu beobachten. Die Reduktionen des maximal benötigten Speicherbedarfs fallen mit dynamischem Umordnen für TLEBDDs aber nicht so groß aus, wie bei den im vorhergehenden Abschnitt für BDDs präsentierten experimentellen Ergebnissen. Der Grund für die geringere Reduktion des Speicherbedarfs und die Laufzeiterhöhungen bei TLEBDDs ist die Beschränkung der für das dynamische Umordnen hier erlaubten Variablenordnungen. Bei den Verifikationsexperimenten mit TLEBDDs und dynamischem Umordnen wurden nur die durch das in Abschnitt 8.3 beschriebene Vorgehen möglichen Variablenordnungen zugelassen. Dies sind nicht alle für die Bildberechnung mit TLEBDDs mit dem in Abschnitt 5.3 beschriebenen Algorithmus möglichen Variablenordnungen. Dadurch können nur die mit dieser beschränkten Menge an Variablenordnungen erzielbaren Werte für Speicherbedarf und Laufzeit erreicht werden. Außerdem kann auch die Berechnung der günstigen Variablenordnungen, bei der hier mit größeren Gruppen von Variablen gearbeitet wird, aufwendiger sein. Die Verwendung der hier erlaubten Variablenordnungen und der zum Umordnen verwendete Ansatz ermöglichen es für kein Verifikationsexperiment dieses mit dynamischem Umordnen erfolgreich abzuschließen, wenn es ohne dynamisches Umordnen nicht erfolgreich abgeschlossen werden konnte. Damit konnten für TLEBDDs nicht die für BDDs durch dynamisches Umordnen beobachteten Vorteile erzielt werden.

8.5.2.3 Vorwärtserreichbarkeitsanalyse mit ETLEBDDs

Mit ETLEBDDs und dynamischem Umordnen der Variablenordnung wurden Verifikationsexperimente zur Vorwärtserreichbarkeitsanalyse mit beiden in Abschnitt 5.8 für ETLEBDDs vorgestellten Explorationsstrategien, *ETLEBDD einfache simultane Exploration* und *ETLEBDD mehrfache simultane Exploration*, durchgeführt. Die Ergebnisse dieser Verifikationsexperimente sind in Tabelle 8.19 dargestellt. Für beide Explorationsstrategien ergibt sich mit dynamischem Umordnen der Variablenordnung für jedes Verifikationsexperiment eine schlechtere Laufzeit, als für das entsprechende Verifikationsexperiment mit dem gleichen Ansatz ohne dynamisches Umordnen. Beim maximal benötigten Speicherbedarf lassen sich mit dynamischem Umordnen ähnliche Wert wie ohne dynamisches Umordnen, die oft leicht unter den dort erzielten Werten liegen, beobachten. Wie bei TLEBDDs lassen sich die Laufzeitsteigerungen und die geringere Reduktion des maximal benötigten Speicherbedarfs, durch die in Abschnitt 8.3 beschriebene Beschränkung der erlaubten Variablenordnungen und das möglicherweise aufwendigere Umordnen der Variablenordnung erklären.

Tabelle 8.19: Vorwärtserreichbarkeitsanalyse mit dynamischem Umordnen

	ETLEBDD einfache simultane Exploration					ETLEBDD mehrfache simultane Exploration				
Testfall	# BDD Knoten	Zeit	# Bildberechnungen		# Umordnungen	# BDD Knoten	Zeit	# Bildberechnungen		# Umordnungen
			Gesamt	ETLE.				Gesamt	ETLE.	
Mut 100	908,003	3m 50s	10605	105	9	41,440	0s	405	105	2
Mut 200	6,960,244	1h 11m	41205	205	13	141,041	4s	805	205	4
Mut 300	23,120,474	17h 11m	91805	305	14	350,002	16s	1205	305	5
Mut 500	-	> 24h	-	-	-	858,016	43s	2005	505	7
Mut 1000	-	> 24h	-	-	-	3,404,396	3m 3s	4005	1005	7
Mut 1500	-	> 24h	-	-	-	8,484,852	9m 2s	6005	1505	8
MutL 4	144,895	4s	490	98	6	196,875	6s	181	149	7
MutL 5	2,067,746	14m 53s	852	142	10	3,171,316	23m 14s	328	268	11
MutL 6	-	> 24h	-	-	-	-	> 24h	-	-	-
Pet 4	87,853	1s	235	47	6	86,160	1s	207	47	6
Pet 5	601,274	47s	444	74	9	547,926	41s	399	74	9
Pet 6	6,193,914	25m 44s	749	107	12	5,502,625	28m 34s	683	107	12
Pet 7	-	> 24h	-	-	-	-	> 24h	-	-	-
DP 20	2,030,880	7m 40s	861	41	10	318,045	13s	261	41	7
DP 21	3,185,705	14m 40s	924	42	10	496,942	15s	273	42	0
DP 23	8,611,696	1h 52m	1104	46	12	1,284,901	1m 38s	322	46	9
DP 25	23,735,234	9h 0m	1300	50	13	3,198,653	4m 29s	375	50	10
DP 28	-	> 24h	-	-	-	13,183,653	1h 20m	477	57	12
DP 30	-	> 24h	-	-	-	34,452,586	14h 54m	541	61	14
DP 50	-	> 24h	-	-	-	-	> 24h	-	-	-
CCP 8	5,290,372	1h 48m	594	66	14	1,442,648	4m 36s	361	89	11
CCP 9	-	> 24h	-	-	-	5,117,788	1h 42m	442	100	13
CCP 11	-	> 24h	-	-	-	-	> 24h	-	-	-

8.6 Vorwärtserreichbarkeitsanalyse mit Symmetriereduktion

In den nachfolgenden Abschnitten werden Ergebnisse von Verifikationsexperimenten zur Vorwärtserreichbarkeitsanalyse unter Benutzung der dynamischen Symmetriereduktion (siehe Abschnitt 2.4.2.3) präsentiert. Bei den dazu durchgeführten Verifikationsexperimenten wurden Nachfolgerberechnungen jeweils mit den in dieser Arbeit zur kombinierten Berechnung des relationalen Produkts vorgestellten Algorithmen (für BDDs Algorithmus 9, für TLEBDDs Algorithmus 12 und für ETLEBDDs Algorithmus 23) durchgeführt. Die in Abschnitt 8.5.1 vorgestellten experimentellen Ergebnisse zeigen, dass die Verwendung dieser Algorithmen zu deutlich besseren Laufzeiten und zu einem geringeren Speicherbedarf, als die entsprechende sequentielle Berechnung der im relationalen Produkt enthaltenen Operationen, führt. In den mit der dynamischen Symmetriereduktion ausgeführten Verifikationsexperimenten wird die dynamische Berechnung von Repräsentanten, bei einer partitionierten und einer monolithischen Transitionsrelation, direkt nach jeder Nachfolgerberechnung ausgeführt. Wie in Abschnitt 2.4.1.1 beschrieben, können in Systemen verschiedene durch Ansätze zur Symmetriereduktion ausnutzbare Symmetriearten vorhanden sein. In den in der vorliegenden Arbeit verwendeten Testfällen liegt, bis auf beim Testfall *DP*, immer vollständige Symmetrie vor. Beim *DP*-Testfall besteht Rotationssymmetrie zwischen den Komponenten.

Es wurden auch Verifikationsexperimente mit zusätzlicher Ausnutzung von Zustandssymmetrien durchgeführt. Die Ausnutzung von Zustandssymmetrien erfolgte dabei nur für die Testfälle mit vorhandener vollständiger Symmetrie. Dies wurde bei diesen wie in Abschnitt 7.3 für vollständige Symmetrie beschrieben ausgeführt. Die dort vorgestellte Implementierung der Benutzung von Zustandssymmetrien wurde für die Verwendung partitionierter Transitionsrelationen entwickelt. Beschrieben wurde diese in Abschnitt 7.3 für das in Abschnitt 7.1 vorgestellte neue Verfahren zur Vorwärtserreichbarkeitsanalyse mit einer partitionierten Transitionsrelation, unter Ausnutzung von Symmetrien. Beim gewöhnlichen Verfahren der Verwendung einer partitionierten Transitionsrelation aus BDDs oder TLEBDDs für die Vorwärtserreichbarkeitsanalyse, wird in einer kompletten Nachfolgerberechnung eine Nachfolgerberechnung mit jeder Partition der Transitionsrelation durchgeführt. Außerdem wird, im Gegensatz zum in Abschnitt 7.1 vorgestellten Ansatz, nur eine Menge mit unexplorierten und noch weiter zu explorierenden Zuständen für die Nachfolgerberechnungen mit allen Partitionen benutzt. In den mit diesen Transitionsrelationstypen mit zusätzlicher Ausnutzung von Zustandssymmetrien durchgeführten Verifikationsexperimenten werden Zustandssymmetrien ausgenutzt, in dem die Menge der unexplorierten Zustände vor der Nachfolgerberechnung für eine Komponente kopiert wird. Aus der kopierten und anschließend für die Nachfolgerberechnung benutzten Menge von Zuständen, werden dann Zustände mit Zustandssymmetrien zur Komponente mit dem nächsthöheren Index, wie in Abschnitt 7.3 beschrieben, entfernt. Dadurch werden diese Zustände für die aktuelle Komponente, für die für eine Partition der Transitionsrelation als nächstes Nachfolgerberechnungen ausgeführt werden, nicht weiter exploriert. Bei Verwendung von ETLEBDDs werden Zustandssymmetrien auf die gleiche Art und nur

vor Nachfolgerberechnungen mit den TLEBDDs, die hier zur Repräsentation der nichtisomorphen Transitionen benutzt werden, ausgenutzt.

8.6.1 Vorwärtserreichbarkeitsanalyse mit BDDs

Für die Benutzung der dynamischen Symmetriereduktion bei der Vorwärtserreichbarkeitsanalyse und eine Repräsentation der Transitionsrelation durch einen monolithischen BDD werden in Tabelle 8.20 Ergebnisse von Verifikationsexperimenten präsentiert. Experimentelle Ergebnisse bei Verwendung einer partitionierten Transitionsrelation aus BDDs sind in Tabelle 8.21 zu finden. Außerdem sind in Tabelle 8.21 Ergebnisse von Verifikationsexperimenten mit einer partitionierten Transitionsrelation, bei denen zusätzlich Zustandssymmetrien ausgenutzt wurden, aufgeführt.

Ergebnisse von Verifikationsexperimenten zur Vorwärtserreichbarkeitsanalyse mit BDDs und dem Algorithmus zur kombinierten Berechnung des relationalen Produkts wurden in Abschnitt 8.5 der vorliegenden Arbeit vorgestellt. Sowohl bei Benutzung einer monolithischen Transitionsrelation, als auch bei Benutzung einer partitionierten Transitionsrelation, ergeben sich für die Testfälle *Mut*, *MutL*, *CCP* und *MCL* durch die Verwendung der dynamischen Symmetriereduktion gegenüber den dort für die entsprechenden Verifikationsexperimente präsentierten Ergebnissen eine sehr große Reduktion der Laufzeit und der maximal benötigten Anzahl an lebenden BDD-Knoten. Außerdem konnten die Verifikationsexperimente mit dynamischer Symmetriereduktion für diese Testfälle aufgrund des geringeren Speicherbedarfs und der geringeren Laufzeit für eine größere Komponentenanzahl erfolgreich durchgeführt werden. Dies resultiert daraus, dass bei Verwendung der dynamischen Symmetriereduktion während der Durchmusterung des Zustandsraums nur Repräsentanten von unter den vorhandenen Symmetrien äquivalenten Zuständen gespeichert und weiterexploriert werden müssen. Bei Verwendung einer partitionierten Transitionsrelation aus BDDs lassen sich, im Vergleich zu den entsprechenden Verifikationsexperimenten ohne Symmetriereduktion, auch noch für den Testfall *Pet* sehr große Laufzeitverbesserungen und große Reduktionen des maximal benötigten Speicherbedarfs beobachten. Beim einzigen für diesen Testfall aufgeführten erfolgreichen Verifikationsexperiment mit einer monolithischen Transitionsrelation ist die Repräsentation der Transitionsrelation sehr groß und bestimmt dort maßgeblich die zur Vorwärtserreichbarkeitsanalyse maximal benötigte Anzahl an lebenden BDD-Knoten. Die Transitionsrelationsgröße und der Bau der Transitionsrelation wird durch die dynamische Symmetriereduktion nicht beeinflusst, wodurch bei diesem Testfall mit dynamischer Symmetriereduktion bei einer monolithischen Transitionsrelation keine signifikanten Verbesserungen des maximal benötigten Speicherbedarfs zu beobachten sind.

Beim Testfall *DP* ist Rotationssymmetrie zwischen den Komponenten vorhanden. Für diesen ergibt sich bei beiden in diesem Abschnitt betrachteten Arten der Speicherung der Transitionsrelation mit Symmetriereduktion ein sehr großer Anstieg der Laufzeit und auch eine Erhöhung des Speicherbedarfs. Der Grund dafür ist die in diesem Testfall vorhandene Rotationssymmetrie. Für diese ist die in dieser Arbeit bei der dynamischen Symmetriereduktion verwendete Repräsentantenberechnung aufwendiger als bei vollständiger Symmetrie, was vermutlich zum Anstieg der Laufzeit und des Speicherbedarfs führt. Der Anstieg

Tabelle 8.20: Vorwärtserreichbarkeitsanalyse ohne dynamisches Umordnen mit dynamischer Symmetriereduktion

	BDD Monolithisch kombiniert		
Testfall	# BDD Knoten	Zeit	# Bildberechnungen
Mut 200	338,718	2m 1s	202
Mut 300	748,081	8m 59s	302
Mut 500	2,046,621	1h 0m	502
Mut 1000	8,093,148	14h 3m	1002
Mut 1500	-	> 24h	-
MutL 4	80,490	1s	88
MutL 5	585,011	29s	132
MutL 6	5,403,142	9m 53s	185
MutL 7	56,713,013	2h 54m	247
Pet 4	98,669,127	1m 1s	47
Pet 5	Speicherül.	-	-
DP 18	9,344,421	56m 35s	37
DP 20	37,038,999	4h 45m	41
DP 21	77,894,159	8h 11m	42
DP 23	Speicherül.	-	-
CCP 8	372,928	12s	66
CCP 9	667,528	34s	74
CCP 11	1,931,804	2m 41s	90
CCP 15	17,291,787	33m 48s	122
CCP 17	67,002,378	1h 54m	138
CCP 21	Speicherül.	-	-
MCL 22	141,257	1m 9s	158
MCL 25	194,022	2m 44s	179
MCL 30	303,991	8m 17s	214
MCL 40	611,431	46m 55s	284
MCL 60	1,703,733	8h 33m	424
MCL 70	2,561,250	20h 27m	494
MCL 80	-	> 24h	-

der Laufzeit ist, im Vergleich zu den Verifikationsexperimenten ohne Symmetriereduktion, bei Benutzung der dynamischen Symmetriereduktion bei einer partitionierten Transitionsrelation deutlich höher als bei Verwendung einer monolithischen Transitionsrelation. Dies liegt an den für partitionierte Transitionsrelationen nach jeder Nachfolgerberechnung durchzuführenden aufwendigen Repräsentantenberechnungen für die Rotationssymmetrie.

Bei Verwendung einer monolithischen Transitionsrelation ist demgegenüber bei einer vollständigen Bildberechnung nur eine Repräsentantenberechnung nach der Bildberechnung mit der gesamten monolithischen Transitionsrelation notwendig.

Mit einer monolithischen Transitionsrelation wird bei einer Nachfolgerberechnung die Menge der mit der gesamten Transitionsrelation aus allen bisher noch nicht weiter explorierten Zuständen erreichbaren Nachfolgerzustände ermittelt. Die Ausnutzung von Zustandssymmetrien durch Löschen von Zuständen aus der Menge der weiter zu explorierenden Zustände, wie es hier für partitionierte Transitionsrelationen durchgeführt wird, ist daher dort nicht direkt möglich. Der Grund ist, dass jeder Zustand mindestens von einer Komponente einer Zustandssymmetriepartition weiter exploriert werden muss. Daher muss hier jeder Zustand in der Menge der weiter zu explorierenden Zustände erhalten bleiben. In der vorliegenden Arbeit wurden daher Verifikationsexperimente mit dynamischer Symmetriereduktion und der zusätzlichen Ausnutzung von Zustandssymmetrien nur für partitionierte Transitionsrelationen ausgeführt. Die Implementierung der Ausnutzung von Zustandssymmetrien erfolgte dabei wie in Abschnitt 7.3 beschrieben. Ergebnisse von Verifikationsexperimenten mit einer partitionierten Transitionsrelation aus BDDs und der zusätzlichen Ausnutzung von Zustandssymmetrien sind im rechten Teil von Tabelle 8.21 angegeben. Die zusätzliche Ausnutzung von Zustandssymmetrien führt bei den Testfällen *Mut*, *Pet*, *CCP* und *MCL* (hier bis auf das Verifikationsexperiment mit 22 Komponenten) zu weiteren Laufzeitverbesserungen, die für die Testfälle *Mut* und *MCL* sehr groß ausfallen. Tendenziell fallen die Laufzeitverbesserungen bei größerer vorhandener Komponentenanzahl höher aus, was an der bei höheren Komponentenanzahlen größeren Menge an ausnutzbaren Zustandssymmetrien liegt. Der maximal benötigte Speicherbedarf ändert sich bei allen Verifikationsexperimenten durch die zusätzliche Ausnutzung von Zustandssymmetrien nur geringfügig. Beim einzigen weiteren Testfall *MutL*, für den noch Verifikationsexperimente mit Ausnutzung von Zustandssymmetrien gemacht wurden, ergibt sich für alle Verifikationsexperimente nahezu die gleiche Laufzeit wie ohne die zusätzliche Ausnutzung von Zustandssymmetrien. Damit führt die Verwendung von Zustandssymmetrien hier bis auf den Testfall *MutL* für alle Testfälle, für die diese implementiert wurden, zu Laufzeitverbesserungen.

Anschließend werden die für die Benutzung der dynamischen Symmetriereduktion bei einer monolithischen und einer partitionierten Transitionsrelation erzielten Ergebnisse verglichen. Dabei fällt auf, dass bei Verwendung einer partitionierten Transitionsrelation aus BDDs die Laufzeit ohne Symmetriereduktion für die Testfälle *MutL* und *CCP* kürzer ist, mit Symmetriereduktion ist aber die Laufzeit für diese Testfälle bei Verwendung einer monolithischen Transitionsrelation kürzer. Eine Ursache dafür kann die gleichzeitige Exploration einer größeren Zustandsmenge bei Nachfolgerberechnungen mit einer monolithischen Transitionsrelation sein. Dadurch kann die Repräsentantenberechnung bei der dynamischen Symmetriereduktion nach einer Nachfolgerberechnung für eine größere Zustandsmenge gleichzeitig durchgeführt werden. Dies kann dazu führen, dass bei der Repräsentantenberechnung mehr Berechnungen durch das nur einmal notwendige Ausführen gleicher Teilberechnungen eingespart werden können und es scheint hier die gemeinsame Symmetriereduktion für alle mit der monolithischen Transitionsrelation explorierten Zu-

stände vorteilhafter zu sein. Allerdings ergeben sich beim Testfall *Mut* bei Verwendung einer partitionierten Transitionsrelation größere Laufzeitverbesserungen und beim Testfall *Pet* lassen sich nur bei Verwendung einer partitionierten Transitionsrelation Laufzeitverbesserungen erzielen. Beim Testfall *Pet* ist dies vermutlich auf die große monolithische Transitionsrelation zurückzuführen.

Tabelle 8.21: Vorwärtserreichbarkeitsanalyse ohne dynamisches Umordnen mit dynamischer Symmetriereduktion

	BDD Partitioniert kombiniert			BDD Partitioniert kombiniert mit Zustandssymmetrien		
Testfall	# BDD Knoten	Zeit	# Bild-berech-nungen	# BDD Knoten	Zeit	# Bild-berech-nungen
Mut 200	502,437	5m 44s	40400	502,437	3m 7s	40400
Mut 300	1,119,175	28m 15s	90600	1,119,175	13m 11s	90600
Mut 500	3,059,215	3h 23m	251000	3,059,215	1h 21m	251000
Mut 1000	-	> 24h	-	12,126,339	18h 9m	1002000
Mut 1500	-	> 24h	-	-	> 24h	-
MutL 4	92,607	2s	352	93,226	2s	352
MutL 5	539,569	36s	660	545,191	37s	660
MutL 6	4,285,114	12m 44s	1110	4,292,785	12m 50s	1110
MutL 7	38,964,957	3h 37m	1729	38,753,474	3h 36m	1729
Pet 4	332,561	0s	188	332,561	0s	188
Pet 5	976,946	3s	370	976,946	3s	370
Pet 6	2,593,565	25s	642	2,593,565	24s	642
Pet 8	110,560,211	23m 5s	1528	110,560,211	21m 29s	1528
Pet 9	Speicherül.	-	-	Speicherül.	-	-
DP 18	4,960,393	11h 1m	666	-	-	-
DP 20	-	> 24h	-	-	-	-
CCP 8	226,840	22s	528	213,444	18s	528
CCP 9	372,641	52s	666	348,850	43s	666
CCP 11	967,432	3m 47s	990	908,351	3m 6s	990
CCP 15	9,266,296	43m 11s	1830	9,043,183	34m 23s	1830
CCP 17	36,351,281	2h 11m	2346	35,975,430	1h 47m	2346
CCP 21	Speicherül.	-	-	Speicherül.	-	-
MCL 22	106,060	1m 24s	3476	95,305	2m 10s	3476
MCL 25	142,530	5m 3s	4475	127,035	3m 1s	4475
MCL 30	218,199	16m 12s	6420	190,447	9m 45s	6420
MCL 40	429,094	1h 33m	11360	364,463	1h 0m	11360
MCL 60	1,131,921	20h 48m	25440	929,782	13h 21m	25440
MCL 70	-	> 24h	-	-	> 24h	-

8.6.2 Vorwärtserreichbarkeitsanalyse neuer Algorithmus dynamische Symmetriereduktion mit BDDs

In Tabelle 8.22 sind Ergebnisse von Verifikationsexperimenten mit dem in Abschnitt 7.1 vorgestellten Algorithmus zur Benutzung der dynamischen Symmetriereduktion, bei Verwendung einer partitionierten Transitionsrelation aus BDDs angegeben. Die Tabelle enthält dabei Ergebnisse von Verifikationsexperimenten, die ohne (*BDD Partitioniert Neu*) und mit (*BDD Partitioniert Neu mit Zustandssymmetrien*) zusätzlicher Ausnutzung von Zustandssymmetrien durchgeführt wurden. Die experimentellen Ergebnisse zeigen, dass der in Abschnitt 7.1 vorgestellte Algorithmus für alle Testfälle, bis auf den Testfall *MutL*, zu sehr großen Laufzeitverbesserungen gegenüber der in Abschnitt 8.6.1 analysierten Verwendung der dynamischen Symmetriereduktion mit einer partitionierten Transitionsrelation aus BDDs führt (siehe Tabelle 8.21). Der Speicherbedarf des Algorithmus aus Abschnitt 7.1 ist bei den Testfällen *Mut* und *Pet* ähnlich und bei den Testfällen *MutL* und *DP* deutlich höher. Beim Testfall *MCL* ist der Speicherbedarf mit diesem Verfahren bei einem Verifikationsexperiment leicht geringer und bei den anderen Verifikationsexperimenten höher. Nur für den Testfall *CCP* kann für alle Verifikationsexperimente ein geringerer Speicherbedarf beobachtet werden. Durch die großen Laufzeitverbesserungen können mit dem Ansatz aus Abschnitt 7.1 aber für die Testfälle *Mut*, *DP* und *MCL* Verifikationsexperimente mit größerer Komponentenanzahl innerhalb der vorgegeben Zeitschranke von 24 Stunden erfolgreich beendet werden.

Die zusätzliche Ausnutzung von Zustandssymmetrien, wie sie in Abschnitt 7.3 beschrieben wurde, führt für die Testfälle *Mut* und *MCL* zu weiteren sehr großen Laufzeitverbesserungen. Bei Benutzung des im letzten Abschnitt behandelten Verfahrens für partitionierte Transitionsrelationen, konnten mit Zustandssymmetrien auch noch für den Testfall *Pet* kleine und für den Testfall *CCP* größere Laufzeitverbesserungen beobachtet werden. Beim Verfahren aus Abschnitt 7.1 bleibt die Laufzeit hier mit Zustandssymmetrien gleich oder nahezu gleich. Für den Testfall *MutL* ergibt sich eine geringe Erhöhung der Laufzeit. Dieser Effekt kann eintreten, wenn der Aufwand zur Erkennung und Ausnutzung von Zustandssymmetrien größer ist, als die dadurch erzielbaren Einsparungen. Die benötigte maximale Anzahl an lebenden BDD-Knoten ist mit Zustandssymmetrien ähnlich der bei den entsprechenden Verifikationsexperimenten ohne Zustandssymmetrien erzielten Knotenanzahl. Gründe für die unterschiedliche Wirkung der Ausnutzung von Zustandssymmetrien bei Testfällen mit diesen beiden Verfahren, können die unterschiedlichen Strategien zur Exploration des Zustandsraums sein. Es werden Zustandssymmetrien im Ansatz aus Abschnitt 7.1 nur nach einer vollständigen Exploration aller mit Bildberechnungen einer Komponente ermittelbaren Zustände berechnet. Beim im letzten Abschnitt für partitionierte Transitionsrelationen behandelten Ansatz werden diese stattdessen nach jeder Bildberechnung berechnet.

Tabelle 8.22: Vorwärtserreichbarkeitsanalyse ohne dynamisches Umordnen mit dynamischer Symmetriereduktion

	BDD Partitioniert Neu			BDD Partitioniert Neu mit Zusandssymmetrien		
Testfall	# BDD Knoten	Zeit	# Bild-berech-nungen	# BDD Knoten	Zeit	# Bild-berech-nungen
Mut 200	542,822	14s	600	542,822	4s	403
Mut 300	1,210,058	1m 0s	900	1,210,058	16s	603
Mut 500	3,310,698	4m 56s	1500	3,310,698	1m 12s	1003
Mut 1000	13,130,320	42m 24s	3000	13,130,320	9m 50s	2003
Mut 1500	29,488,404	2h 32m	4500	29,488,404	34m 13s	3003
MutL 4	235,870	3s	295	237,732	3s	295
MutL 5	2,124,593	1m 7s	562	2,146,053	1m 11s	562
MutL 6	21,899,884	21m 13s	943	22,287,131	22m 29s	943
MutL 7	Speicherül.	-	-	Speicherül.	-	-
Pet 4	332,561	0s	154	332,561	0s	154
Pet 5	976,946	1s	301	976,946	1s	301
Pet 6	2,642,708	9s	533	2,607,439	9s	533
Pet 8	110,560,211	8m 6s	1303	110,560,211	8m 3s	1307
Pet 9	Speicherül.	-	-	Speicherül.	-	-
DP 18	10,085,996	3h 26m	271	-	-	-
DP 20	48,541,083	19h 30m	311	-	-	-
DP 21	-	> 24h	-	-	-	-
CCP 8	117,148	2s	349	111,924	2s	346
CCP 9	192,224	4s	455	184,653	4s	454
CCP 11	556,827	16s	735	544,104	18s	735
CCP 15	7,897,031	2m 53s	1613	7,897,031	2m 53s	1613
CCP 17	35,431,359	8m 28s	2242	35,431,359	8m 21s	2242
CCP 21	Speicherül.	-	-	Speicherül.	-	-
MCL 22	94,327	18s	1499	92,482	14s	1479
MCL 25	132,900	42s	1909	130,516	28s	1886
MCL 30	221,515	2m 6s	2703	217,986	1m 21s	2675
MCL 40	520,084	11m 36s	4703	513,789	7m 42s	4665
MCL 60	1,908,067	2h 7m	10353	1,894,124	1h 21m	10295
MCL 70	3,216,049	5h 39m	13935	3,197,085	3h 34m	13935
MCL 80	5,108,169	12h 23m	18125	5,083,422	7h 37m	18125

8.6.3 Vorwärtserreichbarkeitsanalyse mit TLEBDDs

Die in Abschnitt 4 zur Repräsentation der Transitionsrelation von asynchronen nebenläufigen Systemen mit Transitionslokalität vorgestellte Datenstruktur TLEBDD, kann ebenfalls zusammen mit der dynamischen symbolischen Symmetriereduktion benutzt werden. Dazu sind Ergebnisse von Verifikationsexperimenten ohne (*TLEBDD Partitioniert kombiniert*) und mit zusätzlicher Ausnutzung von Zustandssymmetrien (*TLEBDD Partitioniert kombiniert mit Zustandssymmetrien*), für die Vorwärtserreichbarkeitsanalyse mit Verwendung der kombinierten Berechnung des relationalen Produkts mit TLEBDDs (siehe Abschnitt 4.3), in Tabelle 8.23 angegeben. Bei der Benutzung der dynamischen Symmetriereduktion mit TLEBDDs können die gleichen Explorationsalgorithmen wie mit BDDs verwendet werden. Ebenso kann der Einbau der Benutzung der dynamischen Symmetriereduktion wie bei BDDs erfolgen. Die hier vorgestellten Ergebnisse von Verifikationsexperimenten wurden mit dem gewöhnlichen Ansatz zur Vorwärtserreichbarkeitsanalyse, bei dem eine Explorationsrunde aus einer Bildberechnung mit jedem TLEBDD für eine Partition der Transitionsrelation besteht, ausgeführt (siehe Abschnitt 8.6.1 für die Benutzung dieses Ansatzes mit einer partitionierten Transitionsrelation aus BDDs).

Die experimentellen Ergebnisse zeigen, dass mit Verwendung der dynamischen Symmetriereduktion im Vergleich mit den entsprechenden Verifikationsexperimenten bei Verwendung von TLEBDDs ohne Symmetriereduktion (siehe Abschnitt 8.5.1.2), bis auf den Testfall *DP* für alle Testfälle sehr große Laufzeitverbesserungen erzielt werden können. Ebenso können für diese Testfälle sehr große Verbesserungen des Speicherbedarfs beobachtet werden. Bei der Vorwärtserreichbarkeitsanalyse ohne Symmetriereduktion konnten bei Verwendung von Transitionsrelationen aus TLEBDDs statt partitionierter Transitionsrelationen aus BDDs, bis auf den Testfall *Pet* zumeist nur sehr kleine Reduktionen des maximal benötigten Speicherbedarfs beobachtet werden. Mit dynamischer Symmetriereduktion tritt hingegen auch bei den Testfällen *Mut* und *CCP* bei der Verwendung von TLEBDDs eine deutliche Reduktion des maximal benötigten Speicherbedarfs auf. Betrachtet man die experimentellen Ergebnisse zum Bau der Transitionsrelation mit TLEBDDs für die Testfälle *Pet* und *CCP*, so liegt die für diese Testfälle bei der Vorwärtserreichbarkeitsanalyse maximal benötigte Anzahl an BDD-Knoten, jeweils knapp über der zum Bau der Transitionsrelation für diese benötigten Knotenanzahl (siehe Abschnitt 8.4.1). Damit wird die maximale BDD-Knotenanzahl bei diesen Testfällen vermutlich durch das Erzeugen und die Repräsentation der Transitionsrelation bestimmt oder mitbestimmt. Die Reduktion des Speicherbedarfs mit TLEBDDs ist daher bei diesen Testfällen wahrscheinlich auf den deutlich geringeren Speicherbedarf für die Repräsentation der Transitionsrelation durch TLEBDDs zurückzuführen. Die geringfügig abweichenden maximalen Knotenzahlen bei der Vorwärtserreichbarkeitsanalyse im Vergleich zum Bau der Transitionsrelation, werden vermutlich durch die zusätzlich bei diesen Verifikationsexperimenten vor dem Bau der Transitionsrelation erstellten und notwendigen BDDs (wie z.B. ein BDD mit den Anfangszuständen), verursacht. Bei Benutzung von Verfahren zur Symmetriereduktion reduziert sich der zum Speichern von während der Vorwärtserreichbarkeitsanalyse gefundenen Zuständen benötigte Speicherbedarf. Es müssen hier nur noch Repräsentanten von Äquivalenzklassen von Zuständen gespeichert werden. Dadurch kann der maximale Speicher-

Tabelle 8.23: Vorwärtserreichbarkeitsanalyse ohne dynamisches Umordnen mit dynamischer Symmetriereduktion

	TLEBDD Partitioniert kombiniert			TLEBDD Partitioniert kombiniert mit Zustandssymmetrien		
Testfall	# BDD Knoten	Zeit	# Bild-berech-nungen	# BDD Knoten	Zeit	# Bild-berech-nungen
Mut 200	9,455	5m 14s	40400	9,455	3m 3s	40400
Mut 300	14,434	23m 5s	90600	24,434	12m 35s	90600
Mut 500	23,582	3h 2m	251000	23,582	1h 14m	251000
Mut 1000	-	> 24h	-	47,008	16h 1m	1002000
Mut 1500	-	> 24h	-	-	> 24h	-
MutL 4	69,913	1s	352	70,532	1s	352
MutL 5	505,869	34s	660	511,491	34s	660
MutL 6	4,238,802	11m 40s	1110	4,246,473	11m 32s	1110
MutL 7	38,904,339	3h 16m	1729	38,692,856	3h 16m	1729
Pet 4	93,324	0s	188	93,324	0s	188
Pet 5	252,438	2s	370	252,438	2s	370
Pet 6	633,259	21s	642	633,259	20s	642
Pet 8	17,403,449	18m 40s	1528	17,403,449	17m 2s	1528
Pet 9	40,089,387	1h 27m	2178	40,089,387	1h 21m	2178
DP 18	4,953,290	10h 59m	666	-	-	-
DP 20	-	> 24h	-	-	-	-
CCP 8	192,559	19s	528	179,169	17s	528
CCP 9	301,830	46s	666	278,039	40s	666
CCP 11	646,300	3m 45s	990	587,219	2m 57s	990
CCP 15	2,359,310	37m 58s	1830	2,136,197	29m 36s	1830
CCP 17	4,924,667	1h 47m	2346	4,924,667	1h 26m	2346
CCP 21	-	> 24h	-	78,656,571	21h 11m	3570

bedarf von Testfällen, deren maximaler Speicherbedarf vorher durch den zum Abspeichern von Zuständen benötigten Speicherbedarf bestimmt wurde, nun durch den Speicherbedarf für die Transitionsrelation bestimmt oder stärker durch diesen bestimmt werden. Durch das Verringern der Abhängigkeit des maximalen Speicherbedarfs von der Repräsentation der explorierten Zustände, können bei solchen Testfällen bei Verwendung der dynamischen Symmetriereduktion neben den mit TLEBDDs erzielbaren Laufzeitverbesserungen, auch noch größere Speicherreduktionen als mit BDDs erreicht werden. Allerdings werden die Laufzeitvorteile bei der Benutzung von TLEBDDs statt BDDs etwas geringer. Dies kann darauf zurückzuführen sein, dass nun nur noch Nachfolgerberechnungen für symmetriereduzierte Zustände erfolgen. Dadurch fallen die Einsparungen des bei TLEBDDs nicht notwendigen Durchlaufens von Identitätsmustern mit Symmetriereduktion nicht mehr so häufig an. Zusätzlich bestimmt die bei beiden Ansätzen wiederholt durchzuführende Berechnung von symmetriereduzierten Repräsentanten nun mit die Laufzeit von Verifikationsexperimenten. Die Benutzung von TLEBDDs führt aber mit dynamischer Symmetriereduktion weiterhin für jedes Verifikationsexperiment zu geringeren Laufzeiten als die Benutzung von BDDs.

Im Vergleich zur monolithischen Transitionsrelation mit Benutzung der dynamischen Symmetriereduktion, ist die Laufzeit mit TLEBDDs nur für den Testfall *Pet* und für das Verifikationsexperiment mit dem *CCP*-Testfall für 17 Komponenten geringer. Der Speicherbedarf ist aber für alle Testfälle mit TLEBDDs deutlich geringer. Dies kann den Abschluss von Verifikationsexperimenten ermöglichen, die mit einer monolithischen Transitionsrelation aus einem BDD nicht erfolgreich abgeschlossen werden können (siehe z.B. *Pet*-Testfall). Die neue Strategie zur Benutzung von partitionierten Transitionsrelationen mit BDDs liefert deutlich geringere Laufzeiten als die Benutzung von TLEBDDs. Diese könnte aber auch bei Verwendung von TLEBDDs benutzt werden, wofür in der vorliegenden Arbeit aber keine experimentellen Ergebnisse präsentiert werden. Beim hier vorgestellten Ansatz mit TLEBDDs ergibt sich im Vergleich mit der Anwendung des neuen Verfahrens für BDDs bis auf die Testfälle *MutL* und *CCP* eine deutliche Reduktion des maximal benötigten Speicherbedarfs. Die zusätzliche Ausnutzung von Zustandssymmetrien bei der Verwendung von TLEBDDs, führt für die Testfälle *Mut*, *Pet* und *CCP* zu weiteren teilweise sehr groß ausfallenden Laufzeitverbesserungen

8.6.4 Vorwärtserreichbarkeitsanalyse mit ETLEBDDs

Die dynamische Symmetriereduktion kann auch bei Verwendung der beiden in Abschnitt 5.8 für die Erreichbarkeitsanalyse mit ETLEBDDs vorgestellten Strategien benutzt werden. Dazu sind in Tabelle 8.24 Ergebnisse von Verifikationsexperimenten mit der *einfachen simultanen Exploration* und in Tabelle 8.25 mit der *mehrfachen simultanen Exploration* zu finden. Die Repräsentantenberechnung bei der dynamischen Symmetriereduktion wurde dabei immer direkt nach den Bildberechnungen mit dem ETLEBDD, bzw. für die nichtisomorphen Transitionen unmittelbar nach den Bildberechnungen mit den TLEBDDs, durchgeführt.

Bei Benutzung der Explorationsstrategie *einfache simultane Exploration* lässt sich mit zusätzlicher Benutzung der dynamischen Symmetriereduktion, im Vergleich mit den ent-

sprechenden Verifikationsexperimenten ohne Benutzung der dynamischen Symmetriereduktion, für alle Testfälle bis auf den Testfall *DP* eine deutlich geringere Laufzeit und eine sehr große Reduktion des maximal benötigten Speicherbedarfs erzielen. Für den Testfall *DP* ist eine deutliche Laufzeiterhöhung und ein großer Anstieg des Speicherbedarfs zu beobachten. Der Laufzeitanstieg und auch der Anstieg des Speicherbedarfs beim *DP*-Testfall, lassen sich auf die aufwendigere Anwendung der dynamischen Symmetriereduktion für die bei diesem Testfall vorhandene Rotationssymmetrie zurückführen. Zustandssymmetrien wurden bei Verwendung zusammen mit ETLEBDDs nur für die Bildberechnungen mit den TLEBDDs mit den nicht-isomorphen Transitionen ausgenutzt. Durch die zusätzliche Ausnutzung von Zustandssymmetrien lassen sich für die Testfälle *Pet* und *CCP*, bei gleichem Speicherbedarf, weitere Laufzeitverbesserungen erzielen. Diese fallen jedoch für den *Pet*-Testfall gering aus. Geringe Laufzeiterhöhungen können dagegen für die Testfälle *Mut* und *MutL* bei Ausnutzung von Zustandssymmetrien beobachtet werden.

Bei der Explorationsstrategie der *mehrfachen simultanen Exploration* führt die Benutzung der dynamischen Symmetriereduktion für alle Testfälle, bis auf die Testfälle *Mut* und *DP*, zu sehr großen Laufzeitverbesserungen. Für den *Mut*-Testfall und den *DP*-Testfall liefert hingegen die Anwendung dieser Explorationsstrategie ohne die dynamische Symmetriereduktion deutlich geringere Laufzeiten. Eine große Reduktion der maximal benötigten Anzahl an lebenden BDD-Knoten lässt sich, bis auf den Testfall *DP*, für alle Testfälle beobachten. Beim Testfall *DP* ist die Ursache für die schlechteren Ergebnisse mit Benutzung der dynamischen Symmetriereduktion wieder die aufwendigere Anwendung der dynamischen Symmetriereduktion, bei der dort vorhandenen Rotationssymmetrie. Beim *Mut*-Testfall besitzen die Komponenten nur eine geringe Anzahl an lokalen Zuständen und es können Verifikationsexperimente mit einer großen Anzahl an Komponenten erfolgreich durchgeführt werden. Bei einer großen Komponentenanzahl können bei der Benutzung der dynamischen Symmetriereduktion zur Ermittlung der symmetriereduzierten Repräsentanten eine große Anzahl an Vertauschungen von Komponentenzuständen notwendig sein. Dadurch führt die Anwendung der dynamischen Symmetriereduktion hier vermutlich zu einem höheren Zeitaufwand. Es scheint bei diesem Testfall besser für die Laufzeit zu sein, die bei Bildberechnungen mit dem ETLEBDD neu gefundenen Zustände mit dem in der vorliegenden Arbeit vorgestellten Algorithmus zur simultanen Berechnung des relationalen Produkts mit einem ETLEBDD für mehrere Komponenten, ohne Symmetriereduktion direkt weiter zu explorieren. Dadurch fällt der Zeitaufwand für die bei der dynamischen Symmetriereduktion durchzuführenden Repräsentantenberechnungen nicht an. Die zusätzliche Ausnutzung von Zustandssymmetrien führt bei dieser Explorationsstrategie bei gleichem Speicherbedarf kaum zu Laufzeitveränderungen. Dies liegt vermutlich an der hier nur für die Bildberechnungen mit dem TLEBDD verwendeten Ausnutzung von Zustandssymmetrien.

Vergleicht man das Verfahren der *einfachen simultanen Exploration* mit dem der *mehrfachen simultanen Exploration* bei Benutzung der dynamischen Symmetriereduktion, so führt die mehrfache simultane Exploration weiterhin bei fast allen Verifikationsexperimenten zu besseren und meistens zu deutlich geringeren Laufzeiten. Allerdings fallen die Laufzeitunterschiede bei manchen Testfällen nicht mehr so groß wie beim Vergleich ohne

Symmetriereduktion aus. Beim Speicherbedarf ergab sich bis auf die Testfälle *Mut* und *MutL*, bei denen die *einfache simultane Exploration* manchmal einen geringeren Speicherbedarf besitzt, mit der *mehrfachen simultanen Exploration* für alle Testfälle ein geringerer oder der gleiche Speicherbedarf wie mit der *einfachen simultanen Exploration*.

Tabelle 8.24: Vorwärtserreichbarkeitsanalyse ohne dynamisches Umordnen mit dynamischer Symmetriereduktion

	ETLEBDD einfache simultane Exploration				ETLEBDD einfache simultane Exploration mit Zustandssymmetrien			
Testfall	# BDD Knoten	Zeit	# Bildberechnungen		# BDD Knoten	Zeit	# Bildberechnungen	
			Gesamt	ETLE.			Gesamt	ETLE.
Mut 200	9,264	2m 59s	40602	202	9,264	3m 5s	40602	202
Mut 300	14,115	12m 25s	90902	302	14,115	12m 31s	90902	302
Mut 500	22,703	1h 3m	251502	502	22,703	1h 14m	251502	502
Mut 1000	45,249	15h 54m	1003002	1002	45,249	15h 55m	1003002	1002
Mut 1500	-	> 24h	-	-	-	> 24h	-	-
MutL 4	73,372	1s	440	88	73,372	1s	440	88
MutL 5	574,533	28s	792	132	574,533	30s	792	132
MutL 6	5,396,727	9m 41s	1295	185	5,396,727	10m 33s	1295	185
MutL 7	56,788,736	2h 51m	1976	247	56,788,736	2h 59m	1976	247
Pet 4	93,312	0s	235	47	93,312	0s	235	47
Pet 5	252,094	2s	444	74	252,094	2s	444	74
Pet 6	632,843	21s	749	107	632,843	20s	749	107
Pet 8	17,403,428	18m 28s	1719	191	17,403,428	17m 5s	1719	191
Pet 9	40,087,738	1h 28m	2420	242	40,087,738	1h 21m	2420	242
DP 18	4,969,225	10h 50m	703	37	-	-	-	-
DP 20	-	> 24h	-	-	-	-	-	-
CCP 8	174,251	15s	594	66	174,251	15s	594	66
CCP 9	274,430	33s	740	74	274,430	34s	740	74
CCP 11	595,838	2m 30s	1080	90	595,838	2m 21s	1080	90
CCP 15	2,316,265	26m 41s	1952	122	2,316,265	25m 1s	1952	122
CCP 17	4,922,766	1h 21m	2484	138	4,922,766	1h 16m	2484	138
CCP 21	78,653,868	22h 43m	3740	170	78,653,868	20h 48m	3740	170

Tabelle 8.25: Vorwärtserreichbarkeitsanalyse ohne dynamisches Umordnen mit dynamischer Symmetriereduktion

	ETLEBDD mehrfache simultane Exploration				ETLEBDD mehrfache simultane Exploration mit Zustandssymmetrien			
Testfall	# BDD Knoten	Zeit	# Bildberechnungen		# BDD Knoten	Zeit	# Bildberechnungen	
			Gesamt	ETLE.			Gesamt	ETLE.
Mut 200	11,253	6s	602	202	11,253	6s	602	202
Mut 300	17,106	20s	902	302	17,106	20s	902	302
Mut 500	27,694	1m 26s	1502	502	27,694	1m 26s	1502	502
Mut 1000	55,242	11m 32s	3002	1002	55,242	11m 32s	3002	1002
Mut 1500	83,840	38m 28s	4502	1502	83,840	38m 20s	4502	1502
MutL 4	107,197	1s	151	123	107,197	1s	151	123
MutL 5	778,420	30s	280	225	778,420	31s	280	225
MutL 6	5,622,522	9m 5s	480	384	5,622,522	9m 13s	480	384
MutL 7	50,832,603	2h 10m	770	616	50,832,603	2h 13m	770	616
Pet 4	93,312	0s	211	47	93,312	0s	211	47
Pet 5	252,094	1s	399	74	252,094	1s	399	74
Pet 6	632,843	17s	683	107	632,843	17s	683	107
Pet 8	17,403,428	14m 35s	1599	191	17,403,428	13m 41s	1599	191
Pet 9	40,087,738	1h 6m	2267	242	40,087,738	1h 2m	2267	242
DP 18	4,073,966	3h 28m	217	37	-	-	-	-
DP 20	16,100,131	20h 18m	281	41	-	-	-	-
DP 21	-	> 24h	-	-	-	-	-	-
CCP 8	61,268	3s	361	89	61,268	2s	361	89
CCP 9	87,738	5s	442	100	87,738	5s	442	100
CCP 11	174,453	19s	628	122	174,453	19s	628	122
CCP 15	1,234,765	3m 18s	1096	166	1,234,765	3m 16s	1096	166
CCP 17	4,922,766	11m 27s	1378	188	4,922,766	11m 22s	1378	188
CCP 21	78,653,868	4h 1m	2038	232	78,653,868	4h 0m	2038	232

8.7 Zusammenfassung der Ergebnisse der Verifikationsexperimente

In diesem Abschnitt werden die in den Abschnitten 8.4, 8.5 und 8.6 präsentierten Ergebnisse von Verifikationsexperimenten zusammenfassend betrachtet. Bei den Verifikationsexperimenten zum Bau der Transitionsrelation ohne dynamisches Umordnen der Variablenordnung (siehe Abschnitt 8.4.1), werden der deutlich geringste Speicherbedarf und die niedrigsten Laufzeiten für alle Testfälle mit der partitionierten Transitionsrelation aus TLEBDDs und mit der Repräsentation der Transitionsrelation mit Verwendung von ETLEBDDs erzielt. Diese beiden Arten der Repräsentation der Transitionsrelation besitzen, bis auf den Testfall *MutL*, einen ähnlichen maximalen Speicherbedarf und eine ähnliche Laufzeit zum Bau der Transitionsrelationen. Für den *MutL*-Testfall kann die Transitionsrelation bei Verwendung von ETLEBDDs bei zwei Verifikationsexperimenten deutlich schneller gebaut werden. Die rein durch TLEBDDs und die mit Verwendung von ETLEBDDs repräsentierten Transitionsrelationen können auch für jeden Testfall, bis auf den *Pet*-Testfall, für eine größere Komponentenanzahl als die durch BDDs repräsentierten Transitionsrelationen erfolgreich gebaut werden. Beim *Pet*-Testfall kann die um Identitätsmuster reduzierte Transitionsrelation aus BDDs für die gleiche Komponentenanzahl, wie mit TLEBDDs oder mit dem ETLEBDD, erfolgreich erzeugt werden. Allerdings sind der benötigte maximale Speicherbedarf und die Laufzeit bei der Verwendung von TLEBDDs oder mit dem ETLEBDD deutlich geringer. Für den Testfall *DP* kann die Transitionsrelation mit allen Ansätzen, bis auf die monolithische Transitionsrelation aus einem BDD, für eine große Komponentenanzahl erfolgreich gebaut werden. Mit dem Testfall *MCL* wurden aufgrund der nicht vorhandenen Transitionslokalität keine Verifikationsexperimente mit Transitionsrelationen aus denen Identitätsmuster entfernt wurden durchgeführt. Bei diesem Testfall liefert der Ansatz *BDD Monolithisch* deutlich die besten Ergebnisse.

Mit dynamischem Umordnen der Variablenordnung liefern Repräsentationen der Transitionsrelation mit TLEBDDs oder mit Verwendung eines ETLEBDDs für die Testfälle *Mut*, *MutL*, *DP* und *CCP* die besten Werte bezüglich Speicherbedarf und Laufzeit zum Bau der Transitionsrelation. Diese beiden Arten zur Repräsentation der Transitionsrelation besitzen bei diesen Verifikationsexperimenten einen ähnlichen maximalen Speicherbedarf und oft ähnliche Laufzeiten zum Bau der Transitionsrelation. Bei manchen Verifikationsexperimenten kann die Transitionsrelation mit Verwendung eines ETLEBDDs deutlich schneller gebaut werden. Für den Testfall *Pet* ergeben sich beim Verfahren *BDD Partitioniert ohne Identitätsmuster* und für den Testfall *MCL* beim Ansatz *BDD Monolithisch* jeweils der deutlich geringste Speicherbedarf und die besten Laufzeiten. Beim *Pet*-Testfall liegen die besseren Ergebnisse mit dem Ansatz für BDDs vermutlich mit an den Beschränkungen für die mit TLEBDDs und ETLEBDDs verwendbaren Variablenordnungen, bzw. dem Aufwand zur Anwendung des verwendeten Verfahrens zur Umordnung der Variablenordnung unter Berücksichtigung von Gruppierungen von Variablen, bei dem hier mit größeren Gruppen von Variablen gearbeitet wird (siehe Abschnitt 8.3). Damit liefern TLEBDDs und ETLEBDDs ohne dynamisches Umordnen für alle Testfälle, für die sie verwendet werden können, die besten Werte bezüglich Speicherbedarf und Laufzeit. Mit

dynamischem Umordnen können mit TLEBDDs und ETLEBDDs, obwohl beim für das Umordnen verwendeten Verfahren mit Gruppierungen von Variablen nicht alle erlaubten Variablenordnungen erzeugt werden können, dennoch mit Ausnahme des Testfalls *Pet* für alle Testfälle, für die sie verwendet werden können, ebenso die besten Ergebnisse erzielt werden.

Im Folgenden werden die Laufzeit und der Speicherbedarf von Verifikationsexperimenten zum Bau der Transitionsrelation, ohne und mit Verwendung des dynamischen Umordnens der Variablenordnung, verglichen. Dabei sieht man, dass der mit TLEBDDs und ETLEBDDs ohne dynamisches Umordnen erzielte Speicherbedarf bei den aufgeführten gleichen Verifikationsexperimenten mit den Testfällen *Mut*, *MutL* und *DP*, bei denen mit dynamischem Umordnen mit TLEBDDs und ETLEBDDs die besten Werte erzielt werden, ähnlich dem dort erzielten Speicherbedarf ist. Allerdings ist die Laufzeit zum Bau der Transitionsrelation, bis auf den Testfall *DP*, ohne dynamisches Umordnen deutlich geringer. Nur beim Testfall *CCP*, bei dem mit dynamischem Umordnen bei Verwendung eines ETLEBDDs die besten Ergebnisse zu beobachten sind, ergibt sich mit dynamischem Umordnen eine deutliche Reduktion des Speicherbedarfs und der benötigten Laufzeit. Wie die experimentellen Ergebnisse zeigen, ist es für Testfälle bei denen ohne und mit dynamischem Umordnen bei der Verwendung von TLEBDDs und ETLEBDDs ein ähnlicher maximaler Speicherbedarf erzielt wird besser, die Ansätze ohne dynamisches Umordnen zu verwenden. Bei diesen fällt die zum Umordnen der Variablenordnung notwendige Laufzeit nicht an, die hier zu keiner signifikanten Reduktion des Speicherbedarfs zum Bau der Transitionsrelation geführt hat. Ohne dynamisches Umordnen kann aus diesem Grund auch die Transitionsrelation für den *MutL*-Testfall mit 4000 Komponenten mit geringer Laufzeit erfolgreich gebaut werden, wohingegen mit dynamischem Umordnen mit einem ETLEBDD die Zeitschranke von 24 Stunden überschritten wird. Allerdings können sich bei mit einer gebauten Transitionsrelation anschließend ausgeführten Verifikationsläufen zur Vorwärtserreichbarkeitsanalyse wieder Vorteile durch die weitere Verwendung des dynamischen Umordnens der Variablenordnung ergeben. Ein Grund wenn mit dynamischem Umordnen keine Verbesserung des Speicherbedarfs für TLEBDDs und ETLEBDDs erzielt werden kann, kann in manchen Fällen das in der vorliegenden Arbeit dafür verwendete Verfahren zur Umordnung der Variablenordnung und die damit vorhandenen Beschränkungen der möglichen Variablenordnungen sein. Zusätzlich ist zu beachten, das die Ergebnisse der Verifikationsexperimente ohne dynamisches Umordnen der Variablenordnung stark von der für diese verwendeten Variablenordnung abhängen können. Für den Testfall *Pet* liefert bei Betrachtung der ohne und mit dynamischem Umordnen erzielten Ergebnisse die Strategie *BDD Partitioniert ohne Identitätsmuster* mit dynamischem Umordnen den besten Speicherbedarf und die geringsten Laufzeiten. Dort werden bei Verifikationsexperimenten mit dynamischem Umordnen der Variablenordnung, wie bei den anderen Verifikationsexperimenten mit BDDs und dynamischem Umordnen, jeweils alle verschachtelten Variablenordnungen zugelassen. In solchen Variablenordnungen sind korrespondierende Variablen für aktuelle Zustände und Nachfolgerzustände immer benachbart (siehe Abschnitt 8.3). Es existiert hier eine größere Auswahl an gültigen Variablenordnungen als es bei der Verwendung von TLEBDDs oder ETLEBDDs und dem in der vorliegenden Arbeit

für diese verwendeten Verfahren zum dynamischen Umordnen der Variablenordnung gibt. Dadurch sind mehr Möglichkeiten für das Finden einer Variablenordnung mit möglichst geringem Speicherbedarf vorhanden. Beim Testfall *MCL* sind Laufzeit und Speicherbedarf ohne und mit dynamischem Umordnen bei der monolithischen Transitionsrelation geringer als bei der partitionierten Transitionsrelation aus BDDs. Dabei ist der Speicherbedarf ohne dynamisches Umordnen geringer als mit dynamischem Umordnen.

Zur Vorwärtserreichbarkeitsanalyse ohne dynamische Symmetriereduktion wurden Ergebnisse von Verifikationsexperimenten ohne dynamisches Umordnen der Variablenordnung in Abschnitt 8.5.1 und mit dynamischem Umordnen in Abschnitt 8.5.2 der vorliegenden Arbeit aufgeführt. Ohne dynamisches Umordnen der Variablenordnung ergibt sich bis auf den Testfall *MCL*, bei dem keine Transitionslokalität vorhanden ist, für nahezu alle Verifikationsexperimente bei Benutzung von ETLEBDDs und der Explorationsstrategie der *mehrfachen simultanen Exploration*, die geringste Laufzeit und der niedrigste Speicherbedarf (eine Ausnahme beim Speicherbedarf ist der Testfall *MutL*, für den bei 4 Komponenten der Ansatz *TLEBDD Teilresultate Alles*, bei 5 Komponenten der Ansatz *BDD Monolithisch Teilresultate Bild* und bei 6 Komponenten der Ansatz *TLEBDD Partitioniert kombiniert* den geringsten Speicherbedarf liefert). Für den Testfall *MCL* führt das Verfahren *BDD Monolithisch Kombination Algorithmen* für alle Testfälle, bis auf die Laufzeit beim Verifikationslauf mit 5 Komponenten (hier liefert der Ansatz *BDD Monolithisch kombiniert* die beste Laufzeit), zur geringsten Laufzeit und dem niedrigsten Speicherbedarf.

Bei Verwendung des dynamischen Umordnens der Variablenordnung bei der Vorwärtserreichbarkeitsanalyse, liefern die Strategien *BDD Monolithisch kombiniert* und *BDD Partitioniert kombiniert* für alle Testfälle, bis auf den Testfall *Mut*, sowohl die geringste Laufzeit als auch den niedrigsten Speicherbedarf. Für den Testfall *Mut* führt die Verwendung des Ansatzes *ETLEBDD mehrfache simultane Exploration* zur besten Laufzeit und zum geringsten Speicherbedarf.

Vergleicht man die Laufzeiten der Vorwärtserreichbarkeitsanalyse ohne und mit dynamischem Umordnen der Variablenordnung, können die besten Laufzeiten für die Testfälle *Mut*, *MutL* und *MCL* ohne dynamisches Umordnen der Variablenordnung erzielt werden. Außerdem können bei den Verfahren ohne dynamisches Umordnen bei diesen Testfällen mehr Verifikationsläufe ohne Erreichen der verwendeten maximalen Zeitschranke von 24 Stunden erfolgreich ausgeführt werden. Bei den Testfällen *Pet*, *DP* und *CCP* ergeben sich hingegen mit dynamischem Umordnen der Variablenordnung viel geringere beste Laufzeiten. Ein Grund dafür können bei den Testfällen *Pet* und *CCP* mit dynamischem Umordnen erhaltene deutlich kleinere Repräsentationen der Transitionsrelation, als ohne dynamisches Umordnen, sein (siehe Abschnitt 8.4 zum alleinigen Bau der Transitionsrelation). Durch kleinere Repräsentationen der Transitionsrelation kann bei Bildberechnungen die kleinere Transitionsrelation verwendet werden, wodurch die wiederholt auszuführenden Bildberechnungen schneller ausgeführt werden können. Sind die Laufzeitvorteile durch Verwendung von durch dynamischem Umordnen erhaltene kleinere BDDs größer, als die beim dynamischen Umordnen zum Ermitteln möglichst günstiger Variablenordnungen benötigten Laufzeiten, resultieren mit dynamischem Umordnen der Variablenord-

nung Laufzeitgewinne. Bei Verifikationsexperimenten ohne dynamischem Umordnen der Variablenordnung können die Laufzeit und der Speicherbedarf allerdings stark von der für diese verwendeten Variablenordnung abhängen. Bezüglich des benötigten Speicherbedarfs ergibt sich bei fast allen Verifikationsexperimenten zur Vorwärtserreichbarkeitsanalyse, die erfolgreich mit dynamischem Umordnen ausgeführt werden konnten, ein geringerer maximal benötigter Speicherbedarf als ohne dynamisches Umordnen. Veränderungen der verwendeten Variablenordnung durch dynamisches Umordnen können sich dabei bei der Erreichbarkeitsanalyse auf die Größe der Entscheidungsdiagramme zur Repräsentation der Transitionsrelation, auf die Größe von Entscheidungsdiagrammen zur Repräsentation von Mengen und auch auf die Größe von Zwischenergebnissen von Operationen mit Entscheidungsdiagrammen auswirken.

Für die Vorwärtserreichbarkeitsanalyse mit Benutzung der dynamischen Symmetriereduktion wurden Verifikationsexperimente ohne dynamisches Umordnen der Variablenordnung ausgeführt. Bei diesen ergibt sich bei den Verifikationsexperimenten für alle Testfälle, bis auf die Testfälle *MutL* und *DP*, die beste Laufzeit mit der Strategie *BDD Partitioniert Neu*. Größere Laufzeitverbesserungen lassen sich für diese Strategie bei den Testfällen *Mut* und *MCL* durch die zusätzliche Ausnutzung von Zustandssymmetrien erzielen. Für den Testfall *MutL* wurden hier die besten Laufzeiten mit den für ETLEBDDs vorgestellten Explorationsstrategien erreicht. Dadurch zeigt sich, dass die Verwendung von ETLEBDDs zusammen mit der dynamischen Symmetriereduktion zu sehr gute Ergebnissen führen kann. Für den Testfall *DP* liefert die monolithische Transitionsrelation die beste Laufzeit. Dies ist vermutlich auf die bei der monolithischen Transitionsrelation bei jeder Exploration mit der gesamten Transitionsrelation durch eine Bildberechnung nur einmal auszuführende Anwendung der dynamischen Symmetriereduktion zurückzuführen. Damit fällt der Aufwand zur Repräsentantenberechnung nur einmal für alle in einer Explorationsrunde mit der monolithischen Transitionsrelation in einer Bildberechnung explorierten Zustände an. Dies ist hier vorteilhaft, da die Repräsentantenberechnung für die beim *DP*-Testfall vorhandene Rotationssymmetrie sehr aufwendig ist. Bezüglich des benötigten Speicherbedarfs liefern die Strategien *einfache simultane Exploration* und *mehrfache simultane Exploration*, bei denen ein ETLEBDD verwendet wird, für die Testfälle *Mut*, *Pet*, *DP* und *CCP* die besten Ergebnisse. Beim Testfall *MutL* kann der niedrigste Speicherbedarf mit TLEBDDs erzielt werden und beim Testfall *MCL* mit dem Ansatz *BDD Partitioniert Neu* und mit einer partitionierten Transitionsrelation aus BDDs. Damit zeigt sich, dass bis auf den Testfall *MCL*, für den die für die vorliegende Arbeit neu entwickelten Datenstrukturen TLEBDD und ETLEBDD nicht benutzt werden können, diese Datenstrukturen auch mit zusätzlicher Anwendung der dynamischen Symmetriereduktion für alle Testfälle den geringsten Speicherbedarf liefern können.

Nachfolgend werden die Ergebnisse von Verifikationsexperimenten zur Vorwärtserreichbarkeitsanalyse ohne und mit dynamischem Umordnen der Variablenordnung und die Verifikationsexperimente mit Verwendung der dynamischen Symmetriereduktion verglichen. Dadurch werden die für die in der vorliegenden Arbeit benutzten Testfälle bezüglich Laufzeit und Speicherbedarf effizientesten Verfahren zur Vorwärtserreichbarkeitsanalyse ermittelt. Die besten Laufzeiten werden für die Testfälle *MutL*, *Pet*, *CCP* und *MCL* mit

Verwendung der dynamischen Symmetriereduktion erzielt. Für die Testfälle *Mut* und *DP* führen Verfahren ohne Symmetriereduktion zur besten Laufzeit. Dabei werden beim Testfall *Mut* ohne dynamisches Verändern der Variablenordnung und für den Testfall *DP* mit dynamischem Verändern der Variablenordnung die besten Ergebnisse erzielt. Beim Speicherbedarf führen die Verfahren ohne dynamische Symmetriereduktion und ohne dynamisches Verändern der Variablenordnung für keinen Testfall zum geringsten Speicherbedarf. Mit Symmetriereduktion wird für die Testfälle *Mut*, *MutL*, *CCP* und *MCL* der niedrigste maximal benötigte Speicherbedarf erzielt. Bei Verwendung des dynamischen Umordnens der Variablenordnung wird dieser bei den Testfällen *Pet* und *DP* beobachtet. Allerdings lässt sich die Vorwärtserreichbarkeitsanalyse beim Testfall *Pet* mit Symmetriereduktion aufgrund der geringeren Laufzeit für mehr Komponenten erfolgreich durchführen. Damit gibt es für jeden Ansatz zur Vorwärtserreichbarkeitsanalyse ein Verfahren, das zu den besten Ergebnissen führen kann. Allerdings lassen sich mit dynamischer Symmetriereduktion für die größte Anzahl an Testfällen die besten Ergebnisse erzielen. Zusätzliche Verbesserungen bei Verwendung der dynamischen Symmetriereduktion können vermutlich bei manchen Testfällen, durch zusätzliches Verwenden des dynamischen Umordnens der Variablenordnung erreicht werden. Weitere Auswertungen und Vergleiche von Ergebnissen von Verifikationsexperimenten sind in den Unterabschnitten der Abschnitte 8.4, 8.5 und 8.6 zu finden.

9 Zusammenfassung und Ausblick

In diesem Kapitel werden zuerst in Abschnitt 9.1 die in den Kapiteln 3 bis 7 vorgestellten Ansätze und die in Kapitel 8 präsentierten Ergebnisse der mit diesen durchgeführten Verifikationsexperimente zusammenfassend betrachtet. Anschließend wird in Abschnitt 9.2 ein Ausblick über mögliche weiterführende Arbeiten gegeben.

9.1 Zusammenfassung

Das Ziel dieser Arbeit war die Entwicklung von Verfahren zur Verbesserung der symbolischen BDD-basierten Modellprüfung asynchroner nebenläufiger Systeme. Dazu wurden neue Datenstrukturen zur Speicherung der Transitionsrelation solcher Systeme mit Transitionslokalität, neue Algorithmen zur effizienten Durchführung von Bildberechnungen und Strategien zur Exploration des Zustandsraums bei Benutzung von präsentierten neuen Ansätzen vorgestellt. Außerdem wurden sowohl ein verbessertes Verfahren zur Benutzung der symbolischen dynamischen Symmetriereduktion mit partitionierten Transitionsrelationen, als auch ein neuer Ansatz zur Ausnutzung von Zustandssymmetrien bei der BDD-basierten Modellprüfung präsentiert. Die Nützlichkeit der in dieser Arbeit vorgestellten neuen Ansätze wurde anschließend durch umfassende experimentelle Evaluation bestätigt und diskutiert.

Zur Repräsentation der Transitionsrelation asynchroner nebenläufiger Systeme mit Transitionslokalität wurden die Datenstrukturen TLEBDD und ETLEBDD entwickelt. Durch Entfernung von Identitätsmustern und dem Verwenden einer reduzierten Variablenmenge bei diesen, sind häufig größere isomorphe Teilgraphen vorhanden, was zusätzlich zur Speicherersparnis durch die Entfernung von Identitätsmustern von BDD-Programmpaketen zur Reduktion des Speicherbedarfs ausgenutzt werden kann. Die Verwendung dieser Datenstrukturen führt ohne die Benutzung von Verfahren zum dynamischen Umordnen der Variablenordnung gegenüber gewöhnlichen BDDs zu sehr großen Reduktionen des zum Bau der Transitionsrelation von Systemen benötigten Speicherbedarfs und der dabei anfallenden Laufzeit. Für Verifikationsexperimente mit dynamischem Umordnen der Variablenordnung mit TLEBDDs und ETLEBDDs, wurde in der vorliegenden Arbeit ein bereits bestehendes Verfahren entsprechend konfiguriert. Mit diesem lassen sich nicht alle für diese möglichen Variablenordnungen erzeugen. Dennoch lassen sich mit TLEBDDs und ETLEBDDs mit dynamischem Umordnen noch bei einigen Verifikationsexperimenten zum Bau der Transitionsrelation, die unter den in der vorliegenden Arbeit evaluierten Verfahren beste Laufzeit und der geringste Speicherbedarf erzielen. Auch ermöglichen die neuen Datenstrukturen den erfolgreichen Bau von Transitionsrelationen für Systeme mit größeren Komponentenanzahlen, als es bei Verwendung von gewöhnlichen BDDs oder von

um Identitätsmuster reduzierten BDDs möglich war. Ein weiterer Vorteil der Speichereinsparnis bei der Benutzung von TLEBDDs und ETLEBDDs ist, dass es dadurch möglich sein kann größere Mengen von Transitionen in Partitionen der Transitionsrelation zu repräsentieren. Dadurch können mit den Partitionen auszuführende Bildberechnungen für eine größere Menge von Transitionen auf einmal durchgeführt werden. Dies kann zum Beispiel bei der Erreichbarkeitsanalyse zu Laufzeitverbesserungen führen.

Zur möglichst effizienten Durchführung von Bildberechnungen wurden in der vorliegenden Arbeit für BDDs und TLEBDDs Algorithmen zur kombinierten Berechnung der im relationalen Produkt zur Bildberechnung enthaltenen Operationen präsentiert. Durch Benutzung dieser Algorithmen für Bildberechnungen bei der Erreichbarkeitsanalyse, konnten gegenüber der sequentiellen Berechnung der Operationen des relationalen Produkts sehr große Laufzeitverringerungen und Reduktionen des Speicherbedarfs erzielt werden. Für TLEBDDs wurde mit dem für diese entworfenen Algorithmus auch die direkte Kombination von TLEBDDs und BDDs, ohne einen TLEBDD vorher mit Anfallen der dazu notwendigen Laufzeit und dem benötigten Speicherbedarf in einen entsprechenden BDD expandieren zu müssen, ermöglicht.

Für die effiziente Ausführung von Bildberechnungen mit ETLEBDDs wurde ein Algorithmus vorgestellt, der simultan Bildberechnungen für alle Komponenten durchführt, für die Transitionen in einem ETLEBDD gespeichert sind. Dieser Algorithmus ermöglicht die Ausnutzung von bei einer Bildberechnung für mehrere Komponenten gleich anfallenden Teilberechnungen, in dem diese nur einmal ausgeführt werden. Zusätzlich wurden verschiedene Strategien zur Erreichbarkeitsanalyse mit ETLEBDDs vorgestellt. Mit ETLEBDDs konnten bei der Vorwärtserreichbarkeitsanalyse sehr große Laufzeitverbesserungen und auch Reduktionen des Speicherbedarfs erzielt werden. Die Benutzung von ETLEBDDs liefert ohne dynamisches Umordnen der Variablenordnung und ohne Symmetriereduktion für nahezu alle in der vorliegenden Arbeit verwendeten Testfälle die geringste Laufzeit und den niedrigsten Speicherbedarf. Auch bei zusätzlichem dynamischen Umordnen der Variablenordnung, oder mit Verwendung der dynamischen Symmetriereduktion, führt die Benutzung von ETLEBDDs bei der Vorwärtserreichbarkeitsanalyse für manche Testfälle noch zu den besten Werten.

Mit einem Verfahren, bei dem ein neuer Ansatz zur Berücksichtigung bereits fertig berechneter Teilresultate verwendet wird, wurde eine weitere für Bildberechnungen verwendbare Vorgehensweise entwickelt. Diese kann für BDDs und TLEBDDs benutzt werden und für beide Datenstrukturen werden in der vorliegenden Arbeit Algorithmen zur Bildberechnung mit diesem Ansatz vorgestellt (Kapitel 6 für BDDs und Kapitel 10 für TLEBDDs). Für deren Benutzung wurden verschiedene Ansätze vorgestellt. Mit diesen können für einige Testfälle weitere Effizienzsteigerungen gegenüber der kombinierten Berechnung des relationalen Produkts erzielt werden. Unter den verschiedenen Vorgehensweisen zur Benutzung der neuen Algorithmen ist auch ein einfaches Verfahren, bei dem versucht wird dynamisch während der Erreichbarkeitsanalyse den momentan für Bildberechnungen effizientesten Ansatz auszuwählen. Mit diesem konnten für alle Testfälle Laufzeiten, die nah an der Laufzeit des jeweils besten Verfahrens bei dessen ausschließlicher Benutzung liegen, erreicht werden. Bei den Verifikationsexperimenten mit einem Testfall ergab sich damit

gegenüber dem besten Verfahren bei ausschließlicher Verwendung sogar eine deutliche Laufzeitverbesserung.

Techniken zur Symmetriereduktion können bei in Systemen vorhandenen Symmetrien eingesetzt werden, um die Durchmusterung des Zustandsraums bei der Modellprüfung auf Repräsentanten von Äquivalenzklassen aus unter den vorhandenen Symmetrien äquivalenten Zuständen zu beschränken. In dieser Arbeit wurde für partitionierte Transitionsrelationen ein neuer Ansatz zur Erreichbarkeitsanalyse mit Benutzung der dynamischen Symmetriereduktion vorgestellt. Außerdem wurde eine neue Methode zur Ausnutzung von Zustandssymmetrien, für die bisher nur für die explizite Modellprüfung Ansätze existierten, für die BDD-basierte symbolische Modellprüfung präsentiert. Die Ergebnisse der mit diesen Neuentwicklungen durchgeführten Verifikationsexperimente bestätigen deren Leistungsfähigkeit. Unter den mit Ausnutzung von Symmetrien durchgeführten Verifikationsexperimenten, die ohne dynamisches Umordnen der Variablenordnung durchgeführt wurden, ergibt sich mit dem neu entwickelten Verfahren für fast alle verwendeten Testfälle die geringste Laufzeit. Ebenso lassen sich damit beim Vergleich aller in der vorliegenden Arbeit untersuchten Ansätze zur Vorwärtserreichbarkeitsanalyse für einige Testfälle die besten Laufzeiten erzielen. Das zusätzliche Ausnutzen von Zustandssymmetrien führt bei einigen Verifikationsexperimenten zu weiteren Laufzeitverbesserungen. Außerdem führen Ansätze mit Benutzung von Symmetriereduktion für viele Verifikationsexperimente zum geringsten Speicherbedarf.

9.2 Ausblick

Zu den in dieser Arbeit präsentierten neuen Ansätzen sind einige Weiterentwicklungen denkbar. Auf diese wird im Folgenden eingegangen. In der vorliegenden Arbeit wurden mit TLEBDDs und ETLEBDDs bereits Verifikationsexperimente mit dynamischem Umordnen der Variablenordnung durchgeführt. Bei Bildberechnungen mit TLEBDDs und ETLEBDDs müssen bestimmte Bedingungen, für die bei diesen und für die beim zur Repräsentation der beteiligten Zustandsmenge verwendeten BDD benutzten Variablenordnungen eingehalten werden. Für die Verifikationsexperimente mit dynamischem Umordnen mit TLEBDDs und BDDs wurde ein Sifting-Algorithmus, bei dem Gruppen von Variablen gemeinsam berücksichtigt werden können, verwendet. Durch Bilden entsprechender Gruppen von Variablen, wie in Abschnitt 8.3 beschrieben, konnten mit diesem die Bedingungen für die Variablenordnungen eingehalten werden. Durch die Verwendung dieses Sifting-Algorithmus und die dazu notwendige Gruppierung von Variablen können aber nicht alle bei Bildberechnungen mit TLEBDDs und ETLEBDDs erlaubten Variablenordnungen erzeugt werden. Außerdem zeigen die experimentellen Ergebnisse, dass bei Verwendung dieses Ansatzes die Laufzeit oft deutlich höher als beim dynamischen Umordnen mit verschachtelten Variablenordnungen und ausschließlich mit BDDs durchgeführten Verifikationsexperimenten ist. Daher wäre hier eine sinnvolle Weiterentwicklung ein effizienterer Algorithmus zum dynamischen Umordnen der Variablenordnung für die Benutzung von TLEBDDs und ETLEBDDs, der möglichst viele für diese erlaubten Variablenordnungen erzeugen kann. Bei Benutzung von ETLEBDDs wurden die zwischen

Komponenten isomorphen und in einem ETLEBDD gespeicherten Transitionen manuell ermittelt. Für die effiziente Verwendung von ETLEBDDs in Modellprüfern wäre es sinnvoll, Ansätze zur automatischen Erkennung und Ausnutzung isomorpher Transitionen zu entwickeln und in Modellprüfer einzubauen.

Zur Benutzung des Algorithmus mit dem neuen Verfahren zur Berücksichtigung bereits fertig berechneter Teilresultate wurden in dieser Arbeit mehrere Verfahren beschrieben. Unter anderem wurde ein einfacher Ansatz präsentiert, der versucht mit Hilfe von Laufzeitmessungen einen möglichst effizienten Ansatz für Bildberechnungen dynamisch auszuwählen. Bei diesem Ansatz wurden einige Konstanten fest ausgewählt. Trotz guter Ergebnisse, könnte ein Algorithmus zur dynamischen Auswahl des für Bildberechnungen besten Verfahrens unter Benutzung von Verfahren aus dem Bereich des maschinellen Lernens zu weiteren Effizienzsteigerungen führen. Dort könnten zum Beispiel die hier verwendeten Konstanten während der Erreichbarkeitsanalyse dynamisch angepasst werden, um noch bessere Ergebnisse erzielen zu können. Ein weiterer Verbesserungsvorschlag wäre die Untersuchung der Kombination des Verfahrens zur kombinierten Berechnung des relationalen Produkts, mit und ohne dem neuen Verfahren zur Berücksichtigung von Teilresultaten, innerhalb eines Algorithmus zur Berechnung des relationalen Produkts. Hierzu könnten wieder Verfahren aus dem Bereich des maschinellen Lernens verwendet werden, um innerhalb einer Bildberechnung dynamisch den besten Ansatz auszuwählen.

Für die dynamische Symmetriereduktion sind momentan nur Abstraktionsfunktionen für die vollständige Symmetrie und die Rotationssymmetrie bekannt. Hier wäre die Entwicklung von Abstraktionsfunktionen für andere Symmetriegruppen nützlich, um die Anwendbarkeit der dynamischen Symmetriereduktion für andere Symmetriegruppen zu erweitern. Interessant wäre auch die Untersuchung der Eignung der hier präsentierten neuen Techniken für eine Ausführung auf Mehrkern-CPUs und bei zu erwartenden Effizienzsteigerungen die Entwicklung von entsprechenden Verfahren.

10 Anhang

10.1 Algorithmus mit neuer Kombination von Teilresultaten für TLEBDDs

In Abschnitt 6.1 der vorliegenden Arbeit wurde für BDDs ein Algorithmus zur Berechnung des relationalen Produkts mit Benutzung eines neuen Verfahrens zur Berücksichtigung bereits fertig berechneter Teilresultate vorgestellt. Dieser Ansatz kann auch bei der Berechnung des relationalen Produkts mit einem TLEBDD verwendet werden. In Abschnitt 4.3 wurde mit Algorithmus 12 ein Algorithmus zur kombinierten Berechnung des relationalen Produkts zwischen einem TLEBDD zur Repräsentation einer Menge von Transitionen und einem BDD zur Repräsentation einer zu explorierenden Zustandsmenge präsentiert. Hier wird ebenfalls für verschachtelte Variablenordnungen eine modifizierte Version dieses Algorithmus, bei der das neue Verfahren zur Berücksichtigung bereits fertig berechneter Teilresultate benutzt wird, eingeführt. Dieser neue Algorithmus ist in Algorithmus 31 angegeben. Er verfährt nach dem bei Operationen mit BDDs üblicherweise verwendeten rekursiven Berechnungsprinzip, bei dem in einem Rekursionsschritt eine Aufspaltung in Teilprobleme aus Kofaktoren für eine Belegung der Top-Variable des Rekursionsschritts mit den beiden Booleschen Werten ausgeführt wird. Die Teilprobleme werden in einem Rekursionsschritt ermittelt und es werden in diesem weitere rekursive Aufrufe des Algorithmus zu deren Lösung durchgeführt, falls kein Terminalfall vorhanden ist und das zu berechnende Ergebnis auch nicht in der Ergebnistabelle gefunden wurde.

Wie auch beim für BDDs präsentierten Algorithmus, bekommt dieser Algorithmus im Vergleich mit dem in Algorithmus 12 zur kombinierten Berechnung des relationalen Produkts mit einem TLEBDD vorgestellten Algorithmus, als zusätzlichen Parameter einen BDD *Partial* mit einem bereits fertig berechneten Teilresultat übergeben. Der Inhalt dieses BDDs wird mit dem in einem Rekursionsschritt des Algorithmus zu berechnenden Resultat des relationalen Produkts vereinigt. Dies erfolgt wieder durch simultanes Traversieren des BDDs *Partial* bei rekursiven Aufrufen des Algorithmus *RelProdPartialTLE()* für Teilprobleme. Dabei wird ähnlich wie im in Abschnitt 6.1 für die kombinierte Berechnung des relationalen Produkts mit Berücksichtigung eines bereits fertig berechneten Teilresultats für BDDs beschriebenen und präsentierten Algorithmus vorgegangen. Für die Nachfolgerberechnung mit einem TLEBDD für eine Komponente i, kann das relationale Produkt mit einem TLEBDD mit Berücksichtigung eines bereits fertig berechneten Teilresultats wie folgt beschrieben werden:

$$\exists \vec{x}[\Delta(IdVar, R(\vec{y})\{\vec{y} \leftarrow \vec{x}_{\pi_i}\}) \wedge S(\vec{x})]\{\vec{x}' \leftarrow \vec{x}\} \vee Partial(\vec{x}) \tag{10.1}$$

Algorithmus 31: Kombinierte Berechnung des relationalen Produkts mit einem TLEBDD. Die Vereinigung von Teilresultaten mit dem Ergebnis noch ausstehender Berechnungen erfolgt durch simultanes Durchlaufen von BDDs mit Teilresultaten.

```
1  RelProdPartialTLE(TLEBDD_BDD R, MappingList mapping, ROBDD S,
2                    Index compIndex, ROBDD Partial)
3  if IsTerminalCase(R, S, mapping) then
4  |  return TerminalCase(R, S, mapping, Partial);
5  else
6  |  if ComputedTableHasEntry(RelProdPartialTLE, R, S, compIndex, Partial)
   |  then
7  |  |  return ComputedTable(RelProdPartialTLE, R, S, compIndex, Partial);
8  |  else
9  |  |  if !IsTerminalVertex(R) then
10 |  |  |  mappedVariable = mapping[VariableRoot(R)];
11 |  |  else
12 |  |  |  mappedVariable = TerminalValue;
13 |  |  targetVarPartial = TargetVariable(Partial);
14 |  |  υ = TopVariable(mappedVariable, VariableRoot(S), targetVarPartial);
15 |  |  if υ ==mappedVariable then
16 |  |  |  w = VariableRoot(R); R_{w_0} = R|_{w=0}; R_{w_1} = R|_{w=1};
17 |  |  else
18 |  |  |  R_{w_0} = R_{w_1} = R;
19 |  |  if IsTargetStateVar(υ) || IsIdentityPatternVariable(υ, compIndex) then
20 |  |  |  z = GetCurrentVariable(υ);
21 |  |  |  if IsTargetStateVar(υ) && !IsIdentityPatternVariable(υ, compIndex)
   |  |  |  then
22 |  |  |  |  υ = z;
23 |  |  |  R_1 = RelProdPartialTLE(R_{w_1}, mapping, S|_{υ=1}, compIndex,
24 |  |  |                          Partial|_{z=1});
25 |  |  |  R_0 = RelProdPartialTLE(R_{w_0}, mapping, S|_{υ=0}, compIndex,
26 |  |  |                          Partial|_{z=0});
27 |  |  |  if R_0 == R_1 then
28 |  |  |  |  return R_1;
29 |  |  |  else
30 |  |  |  |  Result = FindOrAddUniqueTable(z, R_0, R_1);
31 |  |  else
32 |  |  |  R_1 = RelProdPartialTLE(R_{w_1}, mapping, S|_{υ=1}, compIndex, Partial);
33 |  |  |  R_0 = RelProdPartialTLE(R_{w_0}, mapping, S|_{υ=0}, compIndex, R_1);
34 |  |  |  Result = R_0;
35 |  |  InsertComputedTable(RelProdPartialTLE, R, S, compIndex, Partial,
36 |  |                      Result);
37 |  |  return Result;
```

Dabei ist $R(\vec{y})$ der BDD eines TLEBDDs zur Repräsentation einer Menge von Transitionen einer Komponente i, $S(\vec{x})$ ein BDD zur Repräsentation einer Zustandsmenge und *Partial($\vec{x}$)* ein BDD mit einem bereits fertig berechneten Teilresultat (siehe auch Beschreibung des relationalen Produkts für die Nachfolgerberechnung mit einem TLEBDD in Abschnitt 4.3).

Algorithmus 32: Funktion *TerminalCase()*, die bei einem in Algorithmus 31 aufgetretenen Terminalfall den BDD mit dem Ergebnis des Terminalfalls berechnet und zurückgibt.

```
1 TerminalCase(TLEBDD_BDD R, ROBDD S, MappingList mapping,
2                ROBDD Partial)
3 if IsTerminalVertexZero(R) || IsTerminalVertexZero(S) then
4     return Partial;
5 if IsTerminalVertexOne(R) then
6     if OnlyGlobalOrLocalVariablesFollow(VariableRoot(S), mapping) ||
7       IsTerminalVertexOne(S) then
8         return BDDTerminalVertexOne;
```

Der im Folgenden vorgestellte Algorithmus *RelProdPartialTLE()*, der in Algorithmus 31 angegeben ist, führt eine kombinierte Berechnung des relationalen Produkts mit einem TLEBDD unter Berücksichtigung eines bereits fertig berechneten Teilresultats durch. Im Algorithmus wird in einem Rekursionsschritt zuerst wieder geprüft, ob ein Terminalfall der rekursiven Berechnung vorliegt. Die im Algorithmus *RelProdPartialTLE()* benutzten Terminalfälle und deren Rückgabewerte sind in Algorithmus 32 angegeben. Im Unterschied zu den Terminalfällen beim gewöhnlichen Algorithmus zur kombinierten Berechnung des relationalen Produkts mit einem TLEBDD, ist hier noch der BDD *Partial* mit einem bereits berechneten Teilresultat zu berücksichtigen. Wie im entsprechenden Algorithmus für BDDs, ist der BDD *Partial* der Rückgabewert der Terminalfälle bei denen der Wurzelknoten des BDDs R oder der Wurzelknoten des BDDs S ein Terminalknoten für den Booleschen Wert 0 ist (siehe Zeile 3 und 4 in Algorithmus 32). Dies ergibt sich aus der Vereinigung des Inhalts des BDDs *Partial* mit der durch den Terminalknoten für den Booleschen Wert 0 repräsentierten Booleschen Funktion, die der Rückgabewert dieser Terminalfälle im gewöhnlichen relationalen Produkt mit einem TLEBDD ist. Weitere Terminalfälle liegen vor, wenn der Wurzelknoten des BDDs des TLEBDDs in einem Rekursionsschritt ein Terminalknoten für den Booleschen Wert 1 ist. Für diesen Fall werden zwei verschiedene Terminalfälle verwendet. Der erste tritt ein, wenn zusätzlich ab der tatsächlichen Variablen des Wurzelknotens des BDDs S in der Variablenordnung nur noch Variablen zur Kodierung des globalen Zustands des Systems oder Variablen zur Kodierung von lokalen Zuständen der Komponente, für die der TLEBDD gebaut wurde, vorkommen. Beim Zweiten muss zusätzlich der Wurzelknoten des BDDs S ein Terminalknoten für den Booleschen Wert 1 sein. Beide Terminalfälle wurden auch im Algorithmus zur kombinierten Berechnung des relationalen Produkts mit einem TLEBDD aus Abschnitt 4.3 benutzt.

Der Rückgabewert dieser beiden Terminalfälle war dort der Terminalknoten für den Booleschen Wert 1. Dieser repräsentiert eine Boolesche Funktion, die bei allen möglichen Belegungen der Eingabevariablen auf den Funktionswert 1 auswertet. Da der Inhalt des BDDs *Partial* in der durch diesen Terminalknoten repräsentierten Menge bereits enthalten ist, führt die hier zusätzlich durchzuführende Vereinigung mit dem BDD *Partial* zu keiner Veränderung des Rückgabewerts. Daher ist der Terminalknoten für den Booleschen Wert 1 in diesen Fällen auch im hier vorgestellten Algorithmus der korrekte Rückgabewert. Die Korrektheit der Rückgabewerte der beim Algorithmus zur Berechnung des gewöhnlichen relationalen Produkts zwischen einem TLEBDD und einem BDD verwendeten Terminalfälle wird in Abschnitt 4.5 gezeigt. Der letzte in der in Algorithmus 13 für den gewöhnlichen Algorithmus mit einem TLEBDD aufgeführten Funktion *TerminalCase()* benutzte Terminalfall wird beim in diesem Abschnitt vorgestellten Algorithmus nicht mehr verwendet. Dort wurde der BDD S als Resultat eines Rekursionsschritts zurückgegeben. Dieser wäre hier aber noch mit einem vorhandenen BDD *Partial* zur Repräsentation eines Teilresultats zu vereinigen, wodurch noch weitere Berechnungen nötig wären und die Berechnung noch nicht direkt terminiert werden kann.

Liegt in einem Rekursionsschritt kein Terminalfall vor wird geprüft, ob das Ergebnis der im aktuellen Rekursionsschritt auszuführenden Berechnung bereits berechnet wurde und noch in der Ergebnistabelle gespeichert ist. Wie beim Algorithmus für BDDs wird der BDD *Partial* beim Zugriff auf die Ergebnistabelle als zusätzlicher Parameter verwendet, um die Eindeutigkeit von Zugriffen auf die Ergebnistabelle sicherzustellen. Zusätzlich wird hier noch der Index *compIndex* der Komponente, für die im TLEBDD Transitionen gespeichert sind, als Übergabeparameter verwendet. Damit wird verhindert, dass bei gleichen Wurzelknoten R und S aber TLEBDDs für verschiedene Komponenten unterschiedliche Ergebnisse zurückerhalten werden können (siehe auch Abschnitt 4.3 zum Algorithmus zur kombinierten Berechnung des relationalen Produkts mit einem TLEBDD). Wurde das Ergebnis der durchzuführenden Berechnung nicht in der Ergebnistabelle gefunden, wird zuerst die tatsächliche Variable zur reduzierten Variablen des Wurzelknotens des BDDs R des TLEBDDs bestimmt (Zeilen 9 bis 12 in Algorithmus 31). Diese entspricht der in der Zuordnungsliste des TLEBDDs für diese reduzierte Variable gespeicherten tatsächlichen Variablen, falls der Wurzelknoten des BDDs R kein Terminalknoten ist (Zeile 10). Andernfalls wird die Variable *mappedVariable* zur Speicherung der entsprechenden tatsächlichen Variablen auf einen Wert gesetzt, der einen Terminalknoten anzeigt (Zeile 12). Danach wird die Top-Variable des aktuellen Rekursionsschritts bestimmt. Hier ist wie im Algorithmus zwischen BDDs der Wurzelknoten des BDDs *Partial* zusätzlich zu berücksichtigen. Da er ein bereits fertig berechnetes Teilresultat beinhaltet, enthält er nur Knoten für Variablen für aktuelle Zustände. Wie in Abschnitt 6.1 beschrieben, sind Variablen für aktuelle Zustände von Knoten dieses BDDs bei der Ermittlung der Top-Variablen als Variablen für Nachfolgerzustände zu berücksichtigen. Nach der Ermittlung der Top-Variablen eines Rekursionsschritts (Zeile 14) werden die für die weiteren rekursiven Aufrufe zu verwendenden Kofaktoren des BDDs des TLEBDDs bestimmt. Dies erfolgt wie im Algorithmus zur kombinierten Berechnung des relationalen Produkts mit einem TLEBDD (siehe Zeilen 15 bis 18). Dabei sind dafür nur neue Kofaktoren zu berechnen, wenn die der reduzierten

Variablen des Wurzelknotens des BDDs R des TLEBDDs entsprechende tatsächliche Variable der Top-Variablen entspricht (Zeile 16). Ansonsten ist der Wurzelknoten des BDDs R in einem nachfolgenden Rekursionsschritt zu berücksichtigen und der BDD R wird unverändert als Übergabeparameter bei den im aktuellen Rekursionsschritt auszuführenden weiteren rekursiven Aufrufen verwendet.

Bei der Durchführung der weiteren rekursiven Aufrufe des Algorithmus für Teilprobleme, werden wie im in Kapitel 6 für BDDs vorgestellten Algorithmus zwei Fälle unterschieden. Der erste Fall tritt ein, wenn die Top-Variable eines Rekursionsschritts eine Variable für Nachfolgerzustände oder eine Variable für Identitätsmuster zur Kodierung von lokalen Zuständen einer anderen Komponente, als die für die der TLEBDD gebaut wurde, ist (siehe Zeile 19). Es werden in diesem Fall bei den rekursiven Aufrufen des Algorithmus Kofaktoren des BDDs *Partial*, für Belegungen der der Top-Variable entsprechenden Variablen für aktuelle Zustände mit Booleschen Werten, als Übergabeparameter verwendet (siehe Zeile 20, Zeile 23 und Zeile 25). In Zeile 20 wird dazu zuerst die Variable für aktuelle Zustände z zur Top-Variablen v des aktuellen Rekursionsschritts ermittelt. Dies ist notwendig, da im BDD *Partial* nur Knoten für Variablen für aktuelle Zustände vorhanden sind, die in Rekursionsschritten des Algorithmus aber als Variablen für Nachfolgerzustände berücksichtigt werden müssen. Für eine Variable für Nachfolgerzustände ist der Rückgabewert der Funktion *GetCurrentVariable()* die benachbarte und zu dieser korrespondierende Variable für aktuelle Zustände. Ist die übergebene Variable v eine Variable für aktuelle Zustände, gibt die Funktion die Variable v selbst zurück. Im für BDDs in Kapitel 6 präsentierten Algorithmus *RelProductPartial()*, wurden neue Kofaktoren des BDDs *Partial* bei rekursiven Aufrufen des Algorithmus für Teilprobleme nur bei einer Top-Variablen für Nachfolgerzustände als Übergabeparameter verwendet. Hier wird diese Vorgehensweise auch bei einer Top-Variablen angewandt, die eine Variable zur Kodierung von lokalen Zuständen einer anderen Komponente, als der für die der TLEBDD gebaut wurde, ist. Diese ist für die Komponente, für die der TLEBDD gebaut wurde, eine Variable zur Kodierung eines Identitätsmusters. Der Grund für das Vorgehen ist, dass die Aufspaltung in Teilprobleme für Top-Variablen für solche Identitätsmuster im Algorithmus bezüglich der korrespondierenden Variablen für aktuelle Zustände und Nachfolgerzustände gemeinsam erfolgt. In einem TLEBDD sind keine Knoten für Variablen für solche Identitätsmuster vorhanden. Top-Variablen für diese Identitätsmuster können daher nur Variablen von Knoten im BDD für die Zustandsmenge S oder Variablen von Knoten im BDD mit dem bereits berechneten Teilresultat *Partial* sein. Durch Knoten für Variablen für aktuelle Zustände im BDD S erlaubte Werte von Variablen, bestimmen bei solchen Identitätsmustern die bei Nachfolgerzuständen erlaubten Werte der Variablen. Im BDD *Partial*, in dem ein bereits berechnetes Teilresultat gespeichert ist, bestimmen die dort vorhandenen Knoten die bei Nachfolgerzuständen für Variablen erlaubten Werte. In diesem BDD sind ebenfalls nur Knoten für Variablen für aktuelle Zustände vorhanden, die aber im Algorithmus *RelProdPartialTLE()* als Knoten für Variablen für Nachfolgerzustände berücksichtigt werden. Um das korrekte Ergebnis für die Nachfolgerzustände zu erhalten, müssen die für eine Top-Variable eines Identitätsmusters zu berücksichtigenden

Knoten der BDDs S und *Partial* bei der gemeinsamen Betrachtung beider Variablen von Identitätsmustern kombiniert und damit gemeinsam durchlaufen werden.

Bei der Aufspaltung in Teilprobleme für die weiteren rekursiven Aufrufe verfährt der Algorithmus bei einer Variablen für Nachfolgerzustände oder bei einer Variablen eines Identitätsmusters zur Kodierung der Änderungen des lokalen Zustands einer anderen Komponente als Top-Variable (siehe Zeile 19), auf drei verschiedene Arten:

1. Ist die Top-Variable v eine Variable für aktuelle Zustände und eine Variable zur Kodierung eines Identitätsmusters zur Kodierung von lokalen Zuständen einer anderen Komponente als der, für die der TLEBDD gebaut wurde (*!IsTargetStateVar(v)* && *IsIdentityPatternVariable(v, compIndex)*):
   In diesem Fall werden Kofaktoren des BDDs S bezüglich der Top-Variable v und Kofaktoren des BDDs *Partial* ebenfalls bezüglich der Top-Variable v (siehe Zeile 20) als Kofaktoren bei den rekursiven Aufrufen (Zeile 23 und Zeile 25) verwendet. Damit werden die für ein solches Identitätsmuster notwendigen Kofaktoren dieser beiden BDDs, für das gemeinsame Behandeln der beiden korrespondierenden Variablen des Identitätsmusters korrekt berücksichtigt.

2. Ist die Top-Variable v eine Variable für Nachfolgerzustände und eine Variable zur Kodierung eines Identitätsmusters zur Kodierung von lokalen Zuständen einer anderen Komponente als der, für die der TLEBDD gebaut wurde (*IsTargetStateVar(v)* && *IsIdentityPatternVariable(v, compIndex)*):
   In diesem Fall wird in Zeile 20 die zur Top-Variablen für Nachfolgerzustände korrespondierende Variable für aktuelle Zustände ermittelt. Die Variable v wird anschließend in Zeile 22 auf die dadurch ermittelte Variable z gesetzt. Damit werden für die BDDs S und *Partial* Kofaktoren bezüglich der zur Top-Variablen für Nachfolgerzustände korrespondierenden Variablen für aktuelle Zustände bei den rekursiven Aufrufen verwendet (Zeile 23 und Zeile 25). Dadurch werden die für ein solches Identitätsmuster notwendigen Kofaktoren dieser beiden BDDs, für das gemeinsame Behandeln der beiden korrespondierenden Variablen des Identitätsmusters korrekt berücksichtigt.

3. Ist die Top-Variable v eine Variable für Nachfolgerzustände und keine Variable zur Kodierung eines Identitätsmusters zur Kodierung von lokalen Zuständen einer anderen Komponente als der, für die der TLEBDD gebaut wurde (*IsTargetStateVar(v)* && *!IsIdentityPatternVariable(v, compIndex)*):
   In diesem Fall werden Kofaktoren des BDDs S bezüglich der Top-Variablen für Nachfolgerzustände bei den rekursiven Aufrufen verwendet (Zeile 23 und Zeile 25). Da der BDD S unabhängig von Variablen für Nachfolgerzustände ist, wird der BDD S hier selbst als Übergabeparameter bei den beiden rekursiven Aufrufen benutzt. Für den BDD *Partial* werden Kofaktoren bezüglich der zur Top-Variablen korrespondierenden Variablen für aktuelle Zustände als Übergabeparameter bei den rekursiven Aufrufen verwendet. Da Variablen für aktuelle Zustände von Knoten des BDDs *Partial* als Knoten für Variablen für Nachfolgerzustände zu berücksichtigen sind, werden bei den rekursiven Aufrufen bei einem im BDD *Partial* zur Top-Variable

vorhandenen entsprechenden Knoten für aktuelle Zustände Kofaktoren bezüglich der Variablen für aktuelle Zustände verwendet. Ist kein zur Top-Variable korrespondierender Knoten für aktuelle Zustände im BDD *Partial* vorhanden, entsprechen die verwendeten Kofaktoren dem BDD *Partial* selbst.

Sind die durch die rekursiven Aufrufe in diesen Fällen ermittelten Teilergebnisse in einem Rekursionsschritt gleich, ist hier kein neuer Knoten zu bauen und es wird anschließend in all diesen Fällen eines dieser Teilergebnisse als Rückgabewert des Algorithmus zurückgegeben (Zeile 28). Unterscheiden sich die beiden Teilergebnisse, wird ein neuer Knoten für die zur Top-Variable korrespondierende Variable für aktuelle Zustände, der die beiden Teilergebnisse als Söhne besitzt, zurückgegeben (Zeile 30).

Ist die Top-Variable eine Variable für aktuelle Zustände zur Kodierung des lokalen Zustands der Komponente, für die der TLEBDD gebaut wurde, so wird wie beim für BDDs präsentierten Algorithmus *RelProductPartial()* verfahren. Der BDD *Partial* wird beim rekursiven Aufruf für die Kofaktoren für eine Belegung der Top-Variable mit dem Booleschen Wert 1 mit übergeben und berücksichtigt (Zeile 32). Das bei diesem rekursiven Aufruf ermittelte Teilergebnis R_1 wird anschließend beim rekursiven Aufruf für eine Belegung der Top-Variable mit dem Booleschen Wert 0 als zu berücksichtigendes Teilergebnis verwendet (Zeile 33). Das Ergebnis dieses rekursiven Aufrufs ist der korrekte Rückgabewert für einen solchen Rekursionsschritt. Die Korrektheit der im Algorithmus verwendeten Aufspaltung in Teilprobleme und die der Kombination von rekursiv ermittelten Teilergebnissen zum Ergebnis eines Rekursionsschritts, kann ähnlich wie in Satz 6.1 für den für BDDs in Kapitel 6 präsentierten Algorithmus angegeben gezeigt werden.

Literaturverzeichnis

[1] *SPIN Model Checking and Software Verification, 7th International SPIN Workshop*, volume 1885 of *Lecture Notes in Computer Science*. Springer, 2000.

[2] B. Becker A. Hett, R. Drechsler. The dd package puma - an on-line documentation, 1996. `http://ira.informatik.uni-freiburg.de/software/puma/pumamain.html`.

[3] Jean-Raymond Abrial. *The B-book - assigning programs to meanings*. Cambridge University Press, 2005.

[4] S. B. Akers. Binary decision diagrams. *IEEE Trans. Comput.*, 27(6):509–516, June 1978.

[5] R. Alur, R.K. Brayton, T.A. Henzinger, S. Qadeer, and S.K. Rajamani. Partial-order reduction in symbolic state space exploration. In *Computer Aided Verification*, volume 1254 of *Lecture Notes in Computer Science*, pages 340–351. Springer Berlin Heidelberg, 1997.

[6] Anuchit Anuchitanukul, Zohar Manna, and Tomás E. Uribe. Differential bdds. In *Computer Science Today*, pages 218–233. Springer-Verlag, 1995.

[7] Christian Appold. Using state symmetries to speed up symmetry reduction in model checking. In *9th International Workshop on Symmetry and Constraint Satisfaction Problems (SymCon09)*, 2009.

[8] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. The MIT Press, 2008.

[9] Sharon Barner and Orna Grumberg. Combining symmetry reduction and under-approximation for symbolic model checking. In *CAV*, pages 93–106, 2002.

[10] Sharon Barner and Orna Grumberg. Combining symmetry reduction and under-approximation for symbolic model checking. *Formal Methods in System Design*, 27:29–66, 2005.

[11] Sharon Barner and Ishai Rabinovitz. Efficient symbolic model checking of software using partial disjunctive partitioning. In *Correct Hardware Design and Verification Methods*, volume 2860 of *Lecture Notes in Computer Science*, pages 35–50. Springer Berlin Heidelberg, 2003.

[12] Gérard Basler, Matthew Hague, Daniel Kroening, C.-H.Luke Ong, Thomas Wahl, and Haoxian Zhao. Boom: Taking boolean program model checking one step further. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 6015 of *Lecture Notes in Computer Science*, pages 145–149. Springer Berlin Heidelberg, 2010.

[13] Gérard Basler, Michele Mazzucchi, Thomas Wahl, and Daniel Kroening. Symbolic counter abstraction for concurrent software. In *Proceedings of the 21st International Conference on Computer Aided Verification*, CAV '09, pages 64–78, Berlin, Heidelberg, 2009. Springer-Verlag.

[14] Gérard Basler, Michele Mazzucchi, Thomas Wahl, and Daniel Kroening. Context-aware counter abstraction. *Form. Methods Syst. Des.*, 36(3):223–245, September 2010.

[15] I. Beer, S. Ben-David, C. Eisner, D. Geist, L. Gluhovsky, T. Heyman, A. Landver, P. Paanah, Y. Rodeh, G. Ronin, and Y. Wolfsthal. Rulebase: Model checking at ibm. In *Computer Aided Verification*, volume 1254 of *Lecture Notes in Computer Science*, pages 480–483. Springer Berlin Heidelberg, 1997.

[16] I. Beer, S. Ben-David, C. Eisner, and A. Landver. Rulebase: an industry-oriented formal verification tool. In *Design Automation Conference Proceedings 1996, 33rd*, pages 655 –660, jun, 1996.

[17] Gerd Behrmann, Alexandre David, and Kim Guldstrand Larsen. A tutorial on uppaal. In *Formal Methods for the Design of Real-Time Systems, International School on Formal Methods for the Design of Computer, Communication and Software Systems, SFM-RT 2004, Bertinoro, Italy, September 13-18, 2004, Revised Lectures*, volume 3185 of *Lecture Notes in Computer Science*, pages 200–236. Springer, 2004.

[18] M. Ben-Ari. *Principles of Concurrent and Distributed Programming (2nd Edition) (Prentice-Hall International Series in Computer Science)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2006.

[19] M. Ben-Ari, Z. Manna, and A. Pnueli. The temporal logic of branching time. In *POPL '81: Proceedings of the 8th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 164–176. ACM, 1981.

[20] B. Bérard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, Ph. Schnoebelen, and P. McKenzie. *Systems and Software Verification: Model-Checking Techniques and Tools*. Springer, 2001.

[21] Girish Bhat, Rance Cleaveland, and Orna Grumberg. Efficient on-the-fly model checking for ctl. In *Proceedings of the 10th Annual IEEE Symposium on Logic in Computer Science*, LICS '95, pages 388–397, Washington, DC, USA, 1995. IEEE Computer Society.

[22] A. Biere. Abcd: an experimental bdd library, 1998. `http://fmv.jku.at/abcd/index.html`.

[23] Armin Biere, Alessandro Cimatti, Edmund M. Clarke, and Yunshan Zhu. Symbolic model checking without bdds. In *Proceedings of the 5th International Conference on Tools and Algorithms for Construction and Analysis of Systems*, TACAS '99, pages 193–207, London, UK, UK, 1999. Springer-Verlag.

[24] M. Block, C. Gröpl, H. Preuß, H. J. Prömel, and A. Srivastav. Efficient ordering of state variables and transition relation partitions in symbolic model checking. Institute of Informatics, Humboldt University of Berlin, 1997.

[25] Roderick Bloem, In-Ho Moon, Kavita Ravi, and Fabio Somenzi. Approximations for fixpoint computations in symbolic model checking. In *Proceedings of the World Multiconference on Systemics, Cybernetics and Informatics*, 2000.

[26] Stefan Blom and Jaco Pol van de. Symbolic reachability for process algebras with recursive data types. In *Theoretical Aspects of Computing*, volume 5160 of *Lecture Notes in Computer Science*, pages 81–95, Berlin, Germany, August 2008. Springer Verlag.

[27] Beate Bollig, Martin Löbbing, and Ingo Wegener. Simulated annealing to improve variable orderings for obdds. In *Int'l Workshop on Logic Synth*, pages 5–5, 1995.

[28] Beate Bollig and Ingo Wegener. Improving the variable ordering of obdds is np-complete. *IEEE Trans. Comput.*, 45(9):993–1002, September 1996.

[29] Dragan Bosnacki, Dennis Dams, and Leszek Holenderski. A heuristic for symmetry reductions with scalarsets. In *Proceedings of the International Symposium of Formal Methods Europe on Formal Methods for Increasing Software Productivity*, FME'01, pages 518–533, London, UK, UK, 2001. Springer-Verlag.

[30] Dragan Bosnacki, Dennis Dams, and Leszek Holenderski. Symmetric spin. *International Journal on Software Tools for Technology Transfer*, 4:92–106, 2002.

[31] Karl S. Brace, Richard L. Rudell, and Randal E. Bryant. Efficient implementation of a bdd package. In *Proceedings of the 27th ACM/IEEE Design Automation Conference*, DAC '90, pages 40–45, New York, NY, USA, 1990. ACM.

[32] Robert King Brayton, Alberto L. Sangiovanni-Vincentelli, Curtis T. McMullen, and Gary D. Hachtel. *Logic Minimization Algorithms for VLSI Synthesis*. Kluwer Academic Publishers, Norwell, MA, USA, 1984.

[33] Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Comput.*, 35(8):677–691, August 1986.

[34] Randal E. Bryant. On the complexity of vlsi implementations and graph representations of boolean functions with application to integer multiplication. *IEEE Trans. Comput.*, 40(2):205–213, February 1991.

[35] Randal E. Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Comput. Surv.*, 24(3):293–318, September 1992.

[36] J. R. Burch, E. M. Clarke, and D. E. Long. Representing circuits more efficiently in symbolic model checking. In *Proceedings of the 28th ACM/IEEE Design Automation Conference*, DAC '91, pages 403–407, New York, NY, USA, 1991. ACM.

[37] J. R. Burch, E. M. Clarke, and D. E. Long. Symbolic model checking with partitioned transition relations. In *International Conference on Very Large Scale Integration*, pages 49–58, 1991.

[38] J. R. Burch, E. M. Clarke, K. L. Mcmillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 10^20 states and beyond. *Inf. Comput.*, 98(2):142–170, 1992.

[39] Jerry R. Burch, Edmund M. Clarke, David E. Long, Kenneth L. McMillan, and David L. Dill. Symbolic model checking for sequential circuit verification. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 13:401–424, 1994.

[40] Gianpiero Cabodi, Paolo Camurati, and Stefano Quer. Improved reachability analysis of large finite state machines. In *Proceedings of the 1996 IEEE/ACM International Conference on Computer-Aided Design*, ICCAD '96, pages 354–360, Washington, DC, USA, 1996. IEEE Computer Society.

[41] Gianpiero Cabodi, Paolo Camurati, and Stefano Quer. Improving symbolic traversals by means of activity profiles. In *Proceedings of the 36th annual ACM/IEEE Design Automation Conference*, DAC '99, pages 306–311, New York, NY, USA, 1999. ACM.

[42] P.J. Cameron. *Permutation Groups*. Cambridge Univ. Press, 1999.

[43] Hyunwoo Cho, Gary D. Hachtel, Enrico Macii, Bernard Plessier, and Fabio Somenzi. Algorithms for approximate fsm traversal. In *Proceedings of the 30th international Design Automation Conference*, DAC '93, pages 25–30, New York, NY, USA, 1993. ACM.

[44] Ming-Ying Chung, Gianfranco Ciardo, and AndyJinqing Yu. A fine-grained fullness-guided chaining heuristic for symbolic reachability analysis. In *Automated Technology for Verification and Analysis*, volume 4218 of *Lecture Notes in Computer Science*, pages 51–66. Springer Berlin Heidelberg, 2006.

[45] G. Ciardo and A. S. Miner. A data structure for the efficient kronecker solution of gspns. In *Proc. 8th Int. Workshop on Petri Nets and Performance Models*, pages 22–31. IEEE Comp. Soc. Press, 1999.

[46] Gianfranco Ciardo, Gerald Lüttgen, and Radu Siminiceanu. Saturation: an efficient iteration strategy for symbolic state space generation. In *Proc. Tools and Algorithms*

for the Construction and Analysis of Systems (TACAS), LNCS 2031, pages 328–342. Springer-Verlag, 2001.

[47] Gianfranco Ciardo, Robert Marmorstein, and Radu Siminiceanu. The saturation algorithm for symbolic state-space exploration. *International Journal on Software Tools for Technology Transfer*, 8:4–25, 2006.

[48] Gianfranco Ciardo and Andy Jinqing Yu. Saturation-based symbolic reachability analysis using conjunctive and disjunctive partitioning. In *Correct Hardware Design and Verification Methods*, volume 3725 of *Lecture Notes in Computer Science*, pages 146–161. Springer Berlin Heidelberg, 2005.

[49] Gianfranco Ciardo, Yang Zhao, and Xiaoqing Jin. Ten years of saturation: A petri net perspective. In *Transactions on Petri Nets and Other Models of Concurrency V*, volume 6900 of *Lecture Notes in Computer Science*, pages 51–95. Springer Berlin Heidelberg, 2012.

[50] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. Nusmv version 2: An opensource tool for symbolic model checking. In *Proc. International Conference on Computer-Aided Verification (CAV 2002)*, volume 2404 of *LNCS*, Copenhagen, Denmark, July 2002. Springer.

[51] E. Clarke, O. Grumberg, and K. Hamaguchi. Another look at ltl model checking. In *Formal Methods in System Design*, pages 415–427. Springer-Verlag, 1994.

[52] E. M. Clarke and I. A. Draghicescu. Expressibility results for linear-time and branching-time logics. In *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency, School/Workshop*, pages 428–437, London, UK, UK, 1989. Springer-Verlag.

[53] E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Logic of Programs, Workshop*, pages 52–71, London, UK, 1982. Springer-Verlag.

[54] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Program. Lang. Syst.*, 8(2):244–263, April 1986.

[55] E. M. Clarke, R. Enders, T. Filkorn, and S. Jha. Exploiting symmetry in temporal logic model checking. *Formal Methods in System Design*, 9(1-2):77–104, 1996.

[56] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model checking*. MIT Press, 2000.

[57] Edmund M. Clarke, E. Allen Emerson, Somesh Jha, and A. Prasad Sistla. Symmetry reductions in model checking. In *CAV*, pages 147–158, 1998.

[58] O. Coudert, C. Berthet, and J. C. Madre. Verification of synchronous sequential machines based on symbolic execution. In *Proceedings of the international workshop on Automatic verification methods for finite state systems*, pages 365–373, New York, NY, USA, 1990. Springer-Verlag New York, Inc.

[59] O. Coudert and J. C. Madre. The implicit set paradigm: A new approach to finite state system verification. *Formal Methods in System Design*, 6:133–145, 1995.

[60] Olivier Coudert and Jean Christophe Madre. A unified framework for the formal verification of sequential circuits. In *IEEE/ACM International Conference on Computer-Aided Design, ICCAD 1990, Santa Clara, CA, USA, November 11-15, 1990. Digest of Technical Papers*, pages 126–129, 1990.

[61] Olivier Coudert, Jean Christophe Madre, and Christian Berthet. Verifying temporal properties of sequential machines without building their state diagrams. In *Computer-Aided Verification*, volume 531 of *Lecture Notes in Computer Science*, pages 23–32. Springer Berlin Heidelberg, 1991.

[62] Costas Courcoubetis, Moshe Y. Vardi, Pierre Wolper, and Mihalis Yannakakis. Memory efficient algorithms for the verification of temporal properties. In *CAV*, pages 233–242, 1990.

[63] David L. Dill, Andreas J. Drexler, Alan J. Hu, and C. Han Yang. Protocol verification as a hardware design aid. In *IEEE International Conference on Computer Design: VLSI in Computers and Processors*, pages 522–525. IEEE Computer Society, 1992.

[64] A. Donaldson. *Automatic techniques for detecting and exploiting symmetry in model checking*. PhD thesis, University of Glasgow, 2007.

[65] A. Donaldson and A. Miller. A computational group theoretic symmetry reduction package for the spin model checker. In *Proceedings of the 11th international conference on Algebraic Methodology and Software Technology*, AMAST'06, pages 374–380, Berlin, Heidelberg, 2006. Springer-Verlag.

[66] A. Donaldson and A. Miller. Exact and approximate strategies for symmetry reduction in model checking. In *Proceedings of the 14th international conference on Formal Methods*, FM'06, pages 541–556, Berlin, Heidelberg, 2006. Springer-Verlag.

[67] A. Donaldson and A. Miller. Symmetry reduction for probabilistic model checking using generic representatives. In S. Graf and W. Zhang, editors, *Proc. 4th Int. Symp. Automated Technology for Verification and Analysis (ATVA'06)*, volume 4218 of *Lecture Notes in Computer Science*, pages 9–23. Springer, 2006.

[68] A. Donaldson and A. Miller. Extending symmetry reduction techniques to a realistic model of computation. *Electr. Notes Theor. Comput. Sci.*, 185:63–76, 2007.

[69] A. Donaldson and A. Miller. On the constructive orbit problem. *Ann. Math. Artif. Intell.*, 57(1):1–35, 2009.

[70] A. Donaldson, A. Miller, and D. Parker. GRIP: Generic representatives in PRISM. In *Proc. 4th International Conference on Quantitative Evaluation of Systems (QEST'07)*, pages 115–116. IEEE Computer Society, 2007.

[71] A. Donaldson, A. Miller, and D. Parker. Language-level symmetry reduction for probabilistic model checking. In *Proceedings of the 2009 Sixth International Conference on the Quantitative Evaluation of Systems*, QEST '09, pages 289–298, Washington, DC, USA, 2009. IEEE Computer Society.

[72] R. Drechsler, B. Becker, and N. Gockel. Genetic algorithm for variable ordering of obdds. *IEE Proceedings - Computers and Digital Techniques*, 143(6):364 –368, nov 1996.

[73] Rolf Drechsler and Bernd Becker. *Graphenbasierte Funktionsdarstellung - Boolesche und Pseudo-Boolesche Funktionen.* Teubner, 1998.

[74] Rüdiger Ebendt, Görschwin Fey, and Rolf Drechsler. *Advanced BDD optimization.* Springer, 2005.

[75] A. Emerson and T. Wahl. On combining symmetry reduction and symbolic representation for efficient model checking. In *Correct Hardware Design and Verification Methods*, 2003.

[76] A. Emerson and T. Wahl. Dynamic symmetry reduction. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, 2005.

[77] E. A. Emerson and A. P. Sistla. Symmetry and model checking. *Formal Methods in System Design*, 9(1-2):105–131, August 1996.

[78] E. A. Emerson and R. J. Trefler. From asymmetry to full symmetry: New techniques for symmetry reduction in model checking. In *Conference on Correct Hardware Design and Verification Methods*, pages 142–156. Springer, 1999.

[79] E. A. Emerson and T. Wahl. Efficient reduction techniques for systems with many components. *Electronic Notes in Theoretical Computer Science*, 130(0):379 – 399, 2005.

[80] E. Allen Emerson and Edmund M. Clarke. Characterizing correctness properties of parallel programs using fixpoints. In *Proceedings of the 7th Colloquium on Automata, Languages and Programming*, pages 169–181, London, UK, UK, 1980. Springer-Verlag.

[81] E. Allen Emerson and Joseph Y. Halpern. "sometimes" and "not never" revisited: on branching versus linear time temporal logic. *J. ACM*, 33(1):151–178, 1986.

[82] E. Allen Emerson, John W. Havlicek, and Richard J. Trefler. Virtual symmetry reduction. In *Logic in Computer Science (LICS)*, pages 121–131. IEEE Computer Society Press, 2000.

[83] E. Allen Emerson and Chin-Laung Lei. Modalities for model checking: Branching time logic strikes back. *Sci. Comput. Program.*, 8(3):275–306, 1987.

[84] Reinhard Enders, Thomas Filkorn, and Dirk Taubner. Generating bdds for symbolic model checking in ccs. *Distrib. Comput.*, 6(3):155–164, April 1993.

[85] E. Felt, G. York, R. Brayton, and A. Sangiovanni-Vincentelli. Dynamic variable reordering for bdd minimization. In *Design Automation Conference, 1993, with EURO-VHDL '93. Proceedings EURO-DAC '93., European*, pages 130 –135, sep 1993.

[86] Jean-Claude Fernandez, Claude Jard, Thierry Jéron, and César Viho. Using on-the-fly verification techniques for the generation of test suites. In *CAV*, pages 348–359, 1996.

[87] Jean-Claude Fernandez and Laurent Mounier. "on the fly" verification of behavioural equivalences and preorders. In *Proceedings of the 3rd International Workshop on Computer Aided Verification*, CAV '91, pages 181–191, London, UK, UK, 1992. Springer-Verlag.

[88] Jean-Claude Fernandez, Laurent Mounier, Claude Jard, and Thierry Jéron. On-the-fly verification of finite transition systems. *Formal Methods in System Design*, 1:251–273, 1992.

[89] Jean-Christophe Filliâtre. Deductive software verification. *STTT*, 13(5):397–403, 2011.

[90] Nissim Francez. *Fairness*. Texts and monographs in computer science. Springer, 1986.

[91] Hiroshige Fujii, Goichi Ootomo, and Chikahiro Hori. Interleaving based variable ordering methods for ordered binary decision diagrams. In *Proceedings of the 1993 IEEE/ACM International Conference on Computer-Aided Design*, ICCAD '93, pages 38–41, Los Alamitos, CA, USA, 1993. IEEE Computer Society Press.

[92] M. Fujita, H. Fujisawa, and N. Kawato. Evaluation and improvement of boolean comparison method based on binary decision diagrams. In *1988 IEEE International Conference on Computer-Aided Design, ICCAD 1988, Santa Clara, CA, USA, November 7-10, 1988. Digest of Technical Papers*, pages 2 –5, nov 1988.

[93] M. Fujita, H. Fujisawa, and Y. Matsunaga. Variable ordering algorithms for ordered binary decision diagrams and their evaluation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 12(1):6 –12, jan 1993.

[94] Masahiro Fujita, Yusuke Matsunaga, and Taeko Kakuda. On variable ordering of binary decision diagrams for the application of multi-level logic synthesis. In *Proceedings of the conference on European design automation*, EURO-DAC '91, pages 50–54, Los Alamitos, CA, USA, 1991. IEEE Computer Society Press.

[95] Daniel Geist and Ilan Beer. Efficient model checking by automated ordering of transition relation partitions. In David L. Dill, editor, *Computer Aided Verification*, volume 818 of *Lecture Notes in Computer Science*, pages 299–310. Springer Berlin Heidelberg, 1994.

[96] Jaco Geldenhuys and Antti Valmari. Techniques for smaller intermediary bdds. In *CONCUR 2001 – Concurrency Theory*, volume 2154 of *Lecture Notes in Computer Science*, pages 233–247. Springer Berlin Heidelberg, 2001.

[97] Jordan Gergov and Christoph Meinel. Boolean manipulation with free bdds: An application in combinational logic verification. In *IFIP Congress (1)*, pages 309–314, 1994.

[98] R. Gerth. Concise promela reference, 1997. `http://www.spinroot.com/spin/Man/Quick.html`.

[99] Amit Goel, Gagan Hasteer, and Randal E. Bryant. Symbolic representation with ordered function templates. In *Proceedings of the 40th annual Design Automation Conference*, DAC '03, pages 431–435, New York, NY, USA, 2003. ACM.

[100] Shankar G. Govindaraju and David L. Dill. Verification by approximate forward and backward reachability. In *Proceedings of the 1998 IEEE/ACM International Conference on Computer-Aided Design*, ICCAD '98, pages 366–370, New York, NY, USA, 1998. ACM.

[101] Shankar G. Govindaraju, David L. Dill, Alan J. Hu, and Mark A. Horowitz. Approximate reachability with bdds using overlapping projections. In *Proceedings of the 35th annual Design Automation Conference*, DAC '98, pages 451–456, New York, NY, USA, 1998. ACM.

[102] Susanne Graf and Hassen Saidi. Construction of abstract state graphs with PVS. In *Conference on Computer Aided Verification CAV'97, Haifa*, volume 1254 of *LNCS*, June 1997.

[103] Wolfgang Günther and Rolf Drechsler. Bdd minimization by linear transformations. In *Advanced Computer Systems*, pages 525–532, 1998.

[104] Wolfgang Günther and Rolf Drechsler. Linear transformations and exact minimization of bdds. In *Great Lakes Symp. VLSI*, pages 325–330, 1998.

[105] V. Gyuris, A. P. Sistla, and E. A. Emerson. On-the-fly model checking under fairness that exploits symmetry. In *Computer-Aided Verification (CAV)*, pages 232–243. Springer-Verlag, 1997.

[106] Gary D. Hachtel and Fabio Somenzi. *Logic Synthesis and Verification Algorithms*. Kluwer Academic Publishers, Norwell, MA, USA, 1st edition, 2000.

[107] Martijn Hendriks, Gerd Behrmann, Kim Larsen, Peter Niebert, and Frits Vaandrager. Adding symmetry reduction to uppaal. In Kim Guldstrand Larsen and Peter Niebert, editors, *Formal Modeling and Analysis of Timed Systems*, volume 2791 of *Lecture Notes in Computer Science*, pages 46–59. Springer Berlin Heidelberg, 2004.

[108] I. Herstein. *Topics in Algebra*. John Wiley & Sons, 1975.

[109] Andreas Hett, Christoph Scholl, and Bernd Becker. State traversal guided by hamming distance profiles. In *MBMV*, pages 57–66. VDE, 2000.

[110] A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker. PRISM: A tool for automatic verification of probabilistic systems. In H. Hermanns and J. Palsberg, editors, *Proc. 12th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'06)*, volume 3920 of *LNCS*, pages 441–444. Springer, 2006.

[111] G. Holzmann. *The Spin Model Checker: Primer and Reference Manual*. Prentice Hall, 2011.

[112] G. Holzmann and D. Peled. An improvement in formal verification. In *FORTE*, pages 197–211, 1994.

[113] P. Huber, A. M. Jensen, L. O. Jepsen, and K. Jensen. Towards reachability trees for high-level petri nets. In *Advances in Petri Nets 1984*, volume 188 of *Lecture Notes in Computer Science*, pages 215–233. Springer Berlin Heidelberg, 1985.

[114] M. Huth and M. Ryan. *Logic in computer science: modelling and reasoning about systems*. Cambridge University Press, 2004.

[115] C. N. Ip and D. L. Dill. Efficient verification of symmetric concurrent systems. In *IEEE International Conference on Computer Design: VLSI in Computers and Processors*, pages 230–234. IEEE Computer Society, 1993.

[116] C. N. Ip and D. L. Dill. Better verification through symmetry. *Formal Methods in System Design*, 9(1-2):41–75, 1996.

[117] N. Ishiura, H. Sawada, and S. Yajima. Minimization of binary decision diagrams based on exchanges of variables. In *Proceedings of the IEEE International Conference on Computer-Aided Design, ICCAD '91*, pages 472 –475, nov 1991.

[118] Hiroaki Iwashita and Tsuneo Nakata. Forward model checking techniques oriented to buggy designs. In *Proceedings of the 1997 IEEE/ACM International Conference on Computer-Aided Design*, ICCAD '97, pages 400–404, Washington, DC, USA, 1997. IEEE Computer Society.

[119] Hiroaki Iwashita, Tsuneo Nakata, and Fumiyasu Hirose. Ctl model checking based on forward state traversal. In *Proceedings of the 1996 IEEE/ACM International Conference on Computer-Aided Design*, ICCAD '96, pages 82–87, Washington, DC, USA, 1996. IEEE Computer Society.

[120] R. Jacobi, N. Calazans, and C. Trullemans. Incremental reduction of binary decision diagrams. In *IEEE International Symposium on Circuits and Systems, 1991*, pages 3174 –3177 vol.5, jun 1991.

[121] Somesh Jha. *Symmetry and induction in model checking.* PhD thesis, Carnegie Mellon University, 1996.

[122] Tommi Junttila. Symmetry reduction algorithms for data symmetries. Research Report A72, Helsinki University of Technology, Laboratory for Theoretical Computer Science, Espoo, Finland, May 2002.

[123] Tommi Junttila. New orbit algorithms for data symmetries. In *Application of Concurrency to System Design 2004*, pages 175–184. IEEE, 2004.

[124] Alexander Kaiser, Daniel Kroening, and Thomas Wahl. Dynamic cutoff detection in parameterized concurrent programs. In *Computer Aided Verification*, volume 6174 of *Lecture Notes in Computer Science*, pages 645–659. Springer Berlin Heidelberg, 2010.

[125] T. Kam, T. Villa, and R. Brayton. Multi-valued decision diagrams: theory and applications. In *Multiple-Valued Logic*, 1998.

[126] Kevin Karplus. Representing boolean functions with if-then-else dags. Technical report, Santa Cruz, CA, USA, 1988.

[127] Thomas Kropf. *Introduction to Formal Hardware Verification: Methods and Tools for Designing Correct Circuits and Systems.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1st edition, 1999.

[128] M. Kwiatkowska, G. Norman, and D. Parker. Symmetry reduction for probabilistic model checking. In *Proc. 18th International Conference on Computer Aided Verification (CAV'06)*, volume 4114 of *LNCS*, pages 234–248. Springer, 2006.

[129] M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *Proc. 23rd International Conference on Computer Aided Verification (CAV'11)*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011.

[130] L. Lamport. "sometimes" is sometimes "not never". In *Proceedings of the 7th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, POPL '80, pages 174–185, New York, NY, USA, 1980. ACM.

[131] Jean-Louis Lassez, V. L. Nguyen, and Liz Sonenberg. Fixed point theorems and semantics: A folk tale. *Inf. Process. Lett.*, 14(3):112–116, 1982.

[132] C. Y. Lee. Representation of switching circuits by binary-decision programs. *The Bell System Technical Journal*, 38(4):985–999, July 1959.

[133] Flavio Lerda, Nishant Sinha, and Michael Theobald. Symbolic model checking of software. *Electronic Notes in Theoretical Computer Science*, 89(3), 2003.

[134] Michael Leuschel and Michael Butler. Prob: A model checker for b. In *FME 2003: Formal Methods*, volume 2805 of *Lecture Notes in Computer Science*, pages 855–874. Springer Berlin Heidelberg, 2003.

[135] Michael Leuschel, Michael Butler, Corinna Spermann, and Edd Turner. Symmetry reduction for b by permutation flooding. In *Proceedings B'2007*, volume 4355 of *Lecture Notes in Computer Science*, pages 79–93. Springer-Verlag, 2007.

[136] Michael Leuschel and Thierry Massart. Efficient approximate verification of b via symmetry markers. *Proceedings International Symmetry Conference*, 2007.

[137] Michael Leuschel and Thierry Massart. Efficient approximate verification of b and z models via symmetry markers. *Annals of Mathematics and Artificial Intelligence*, 59(1):81–106, 2010.

[138] Orna Lichtenstein and Amir Pnueli. Checking that finite state concurrent programs satisfy their linear specification. In *Proceedings of the 12th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, POPL '85, pages 97–107, New York, NY, USA, 1985. ACM.

[139] Jean-Christophe Madre and Jean-Paul Billon. Proving circuit correctness using formal comparison between expected and extracted behaviour. In *Proceedings of the 25th ACM/IEEE Design Automation Conference*, DAC '88, pages 205–210. IEEE Computer Society Press, 1988.

[140] S. Malik, A. R. Wang, R. K. Brayton, and A. Sangiovanni-Vincentelli. Logic verification using binary decision diagrams in a logic synthesis environment. In *Proceedings of the IEEE International Conference on Computer-Aided Design, 1988. ICCAD-88. Digest of Technical Papers*, pages 6 –9, nov 1988.

[141] B. D. McKay and A. Piperno. *nauty* and *Traces* user's guide (version 2.5), 2013.

[142] Kenneth Lauchlin McMillan. *Symbolic model checking: an approach to the state explosion problem.* PhD thesis, Pittsburgh, PA, USA, 1992.

[143] Christoph Meinel, Fabio Somenzi, and Thorsten Theobald. Linear sifting of decision diagrams. In *Proceedings of the 34th annual Design Automation Conference*, DAC '97, pages 202–207, New York, NY, USA, 1997. ACM.

[144] Christoph Meinel and Thorsten Theobald. *Algorithmen und Datenstrukturen im VLSI-Design: OBDD - Grundlagen und Anwendungen.* Springer, 1998.

[145] J. M. Mellor-Crummey and M. L. Scott. Algorithms for scalable synchronization on shared-memory multiprocessors. *ACM Transactions on Computer Systems*, 9:21–65, 1991.

[146] José Meulen and Charles Pecheur. Combining partial-order reduction and symbolic model checking to verify ltl properties. In *NASA Formal Methods*, volume 6617 of *Lecture Notes in Computer Science*, pages 406–421. Springer Berlin Heidelberg, 2011.

[147] A. Miller, A. Donaldson, and M. Calder. Symmetry in temporal logic model checking. *ACM Computing Surveys (CSUR)*, 38(3):8, 2006.

[148] Shin-ichi Minato, Nagisa Ishiura, and Shuzo Yajima. Shared binary decision diagram with attributed edges for efficient boolean function manipulation. In *Proceedings of the 27th ACM/IEEE Design Automation Conference*, DAC '90, pages 52–57, New York, NY, USA, 1990. ACM.

[149] Andrew S. Miner. Efficient solution of gspns using canonical matrix diagrams. In *Proceedings of the 9th International Workshop on Petri Nets and Performance Models*, pages 101–110. IEEE Comp. Soc. Press, 2001.

[150] Andrew S. Miner. Implicit gspn reachability set generation using decision diagrams. *Perform. Eval.*, 56(1-4):145–165, March 2004.

[151] Andrew S. Miner. Saturation for a general class of models. *IEEE Transactions on Software Engineering*, 32(8):559–570, Aug. 2006.

[152] Paul Molitor and Christoph Scholl. *Datenstrukturen und effiziente Algorithmen für die Logiksynthese kombinatorischer Schaltungen.* Teubner, 1999.

[153] Amit Narayan, Adrian J. Isles, Jawahar Jain, Robert K. Brayton, and Alberto L. Sangiovanni-Vincentelli. Reachability analysis using partitioned-robdds. In *Proceedings of the 1997 IEEE/ACM International Conference on Computer-Aided Design*, ICCAD '97, pages 388–393, Washington, DC, USA, 1997. IEEE Computer Society.

[154] Amit Narayan, Jawahar Jain, M. Fujita, and A. Sangiovanni-Vincentelli. Partitioned robdds - a compact, canonical and efficiently manipulable representation for boolean functions. In *Proceedings of the 1996 IEEE/ACM International Conference on Computer-Aided Design*, ICCAD '96, pages 547–554, Washington, DC, USA, 1996. IEEE Computer Society.

[155] Shipra Panda and Fabio Somenzi. Who are the variables in your neighborhood. In *Proceedings of the 1995 IEEE/ACM International Conference on Computer-Aided Design*, ICCAD '95, pages 74–77, Washington, DC, USA, 1995. IEEE Computer Society.

[156] Shipra Panda, Fabio Somenzi, and Bernard F. Plessier. Symmetry detection and dynamic variable ordering of decision diagrams. pages 628–631, 1996.

[157] Doron Peled. Combining partial order reductions with on-the-fly model-checking. In *Computer Aided Verification*, volume 818 of *Lecture Notes in Computer Science*, pages 377–390. Springer Berlin Heidelberg, 1994.

[158] G. L. Peterson. Myths about the mutual exclusion problem. *Inf. Process. Lett.*, 12(3):115–116, 1981.

[159] Carl Pixley. A theory and implementation of sequential hardware equivalence. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 11(12):1469–1478, 1992.

[160] A. Pnueli. A temporal logic of concurrent programs. *Theoretical Computer Science*, 13:45–60, 1981.

[161] A. Pnueli, S. Ruah, and L. Zuck. Automatic deductive verification with invisible invariants. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 2031 of *Lecture Notes in Computer Science*, pages 82–97. Springer Berlin Heidelberg, 2001.

[162] Amir Pnueli, Jessie Xu, and Lenore D. Zuck. Liveness with (0, 1, ∞)-counter abstraction. In *Computer-Aided Verification (CAV)*, pages 107–122, 2002.

[163] J.-P. Queille and J. Sifakis. Specification and verification of concurrent systems in cesar. In *Proceedings of the 5th Colloquium on International Symposium on Programming*, pages 337–351, London, UK, 1982. Springer-Verlag.

[164] Rajeev K. Ranjan, Adnan Aziz, Robert K. Brayton, Bernard Plessier, and Carl Pixley. Efficient bdd algorithms for fsm synthesis and verification. In *IEEE/ACM Proceedings International Workshop on Logic Synthesis, Lake Tahoe (NV)*, 1995.

[165] Kavita Ravi, Kenneth L. McMillan, Thomas R. Shiple, and Fabio Somenzi. Approximation and decomposition of binary decision diagrams. In *Proceedings of the 35th annual Design Automation Conference*, DAC '98, pages 445–450, New York, NY, USA, 1998. ACM.

[166] Kavita Ravi and Fabio Somenzi. High-density reachability analysis. In *Proceedings of the 1995 IEEE/ACM International Conference on Computer-Aided Design*, ICCAD '95, pages 154–158, Washington, DC, USA, 1995. IEEE Computer Society.

[167] Kavita Ravi and Fabio Somenzi. Hints to accelerate symbolic traversal. In *Correct Hardware Design and Verification Methods*, volume 1703 of *Lecture Notes in Computer Science*, pages 250–266. Springer Berlin Heidelberg, 1999.

[168] Alan Robinson and Andrei Voronkov, editors. *Handbook of automated reasoning.* Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, 2001.

[169] Oriol Roig, Jordi Cortadella, and Enric Pastor. Verification of asynchronous circuits by bdd-based model checking of petri nets. In *Application and Theory of Petri Nets 1995*, volume 935 of *Lecture Notes in Computer Science*, pages 374–391. Springer Berlin Heidelberg, 1995.

[170] J.S. Rose. *A Course in Group Theory.* Dover Publications, 1994.

[171] Kristin Y. Rozier. Survey: Linear temporal logic symbolic model checking. *Comput. Sci. Rev.*, 5(2):163–203, May 2011.

[172] Richard Rudell. Dynamic variable ordering for ordered binary decision diagrams. In *Proceedings of the 1993 IEEE/ACM International Conference on Computer-Aided Design*, ICCAD '93, pages 42–47, Los Alamitos, CA, USA, 1993. IEEE Computer Society Press.

[173] Jagesh Sanghavi, Rajeev K. Ranjan, Robert K. Brayton, and Alberto Sangiovanni-Vincentelli. High performance bdd package by exploiting memory hierarchy. In *Proceedings of the 33rd Annual Design Automation Conference*, DAC '96, pages 635–640, New York, NY, USA, 1996. ACM.

[174] Klaus Schneider. *Verification of Reactive Systems: Formal Methods and Algorithms.* Springer-Verlag, 2004.

[175] Christoph Scholl. *Mehrstufige Logiksynthese unter Ausnutzung funktionaler Eigenschaften.* Dissertation, Universität des Saarlandes, 1996.

[176] Claude Elwood Shannon. A symbolic analysis of relay and switching circuits. *Transactions of the American Institute of Electrical Engineers*, 57(12):713–723, Dec 1938.

[177] Detlef Sieling and Ingo Wegener. Graph driven bdds - a new data structure for boolean functions. *Theoretical Computer Science*, 141(1–2):283 – 310, 1995.

[178] A. P. Sistla, V. Gyuris, and E. A. Emerson. SMC: a symmetry-based model checker for verification of safety and liveness properties. *ACM Trans. Softw. Eng. Methodol.*, 9(2):133–166, 2000.

[179] Marc Solé and Enric Pastor. Traversal techniques for concurrent systems. In *Formal Methods in Computer-Aided Design*, volume 2517 of *Lecture Notes in Computer Science*, pages 220–237. Springer Berlin Heidelberg, 2002.

[180] Fabio Somenzi. Binary decision diagrams. In *Calculational System Design, volume 173 of NATO Science Series F: Computer and Systems Sciences*, pages 303–366. IOS Press, 1999.

[181] Fabio Somenzi. Cudd: Cu decision diagram package, release 2.5.0, 2004. `http://vlsi.colorado.edu/~fabio/CUDD/cuddIntro.html`.

[182] Corinna Spermann and Michael Leuschel. Prob gets nauty: Effective symmetry reduction for b and z models. In *Proceedings of the 2008 2nd IFIP/IEEE International Symposium on Theoretical Aspects of Software Engineering*, TASE '08, pages 15–22, Washington, DC, USA, 2008. IEEE Computer Society.

[183] Alfred Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5(2):285–309, 1955.

[184] H. J. Touati, H. Savoj, B. Lin, R. K. Brayton, and A. Sangiovanni-Vincentelli. Implicit state enumeration of finite state machines using bdds. In *IEEE/ACM International Conference on Computer-Aided Design, ICCAD 1990, Santa Clara, CA, USA, November 11-15, 1990. Digest of Technical Papers*, pages 130–133, 1990.

[185] Richard Trefler and Thomas Wahl. Extending symmetry reduction by exploiting system architecture. In *Proceedings of the 10th International Conference on Verification, Model Checking, and Abstract Interpretation*, VMCAI '09, pages 320–334, Berlin, Heidelberg, 2009. Springer-Verlag.

[186] Edd Turner, Michael Leuschel, Corinna Spermann, and Michael Butler. Symmetry reduced model checking for b. In *First Joint IEEE/IFIP Symposium on Theoretical Aspects of Software Engineering, TASE 2007*, pages 5–8. IEEE Computer Society.

[187] Moshe Y. Vardi and Pierre Wolper. An automata-theoretic approach to automatic program verification (preliminary report). In *LICS*, pages 332–344, 1986.

[188] T. Wahl. Adaptive symmetry reduction. In *Computer-Aided Verification (CAV)*, 2007.

[189] T. Wahl. *Exploiting Replication in Automated Program Verification*. PhD thesis, Austin, TX, USA, 2007.

[190] T. Wahl. *The* Sviss *Symbolic Verifier*. Swiss Federal Institute of Technology, Zürich, `http://web.comlab.ox.ac.uk/people/Thomas.Wahl/Sviss`, 2008.

[191] T. Wahl, N. Blanc, and A. Emerson. Sviss: Symbolic verification of symmetric systems. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, 2008.

[192] Thomas Wahl and Vijay D 'Silva. A lazy approach to symmetry reduction. *Formal Aspects of Computing*, 22:713–733, 2010.

[193] Thomas Wahl and Alastair Donaldson. Replication and abstraction: Symmetry in automated formal verification. *Symmetry*, 2(2):799–847, 2010.

[194] Min Wan and Gianfranco Ciardo. Symbolic state-space generation of asynchronous systems using extensible decision diagrams. In *Proceedings of the 35th Conference on Current Trends in Theory and Practice of Computer Science*, SOFSEM '09, pages 582–594, Berlin, Heidelberg, 2009. Springer-Verlag.

[195] Chao Wang, Zijiang Yang, Franjo Ivančić, and Aarti Gupta. Disjunctive image computation for software verification. *ACM Trans. Des. Autom. Electron. Syst.*, 12(2), April 2007.

[196] Pierre Wolper. Expressing interesting properties of programs in propositional temporal logic. In *Proceedings of the 13th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, POPL '86, pages 184–193, New York, NY, USA, 1986. ACM.

[197] Bwolen Yang, Randal E. Bryant, David R. O'Hallaron, Armin Biere, Olivier Coudert, Geert Janssen, Rajeev K. Ranjan, and Fabio Somenzi. A performance study of bdd-based model checking. In *Proceedings of the Formal Methods in Computer-Aided Design*, pages 255–289. Springer-Verlag, 1998.

[198] Zijiang Yang and Rajeev Alur. Variable reuse for efficient image computation. In *Formal Methods in Computer-Aided Design*, volume 3312 of *Lecture Notes in Computer Science*, pages 430–444. Springer Berlin Heidelberg, 2004.

Abbildungsverzeichnis

Algorithmenverzeichnis

Tabellenverzeichnis