Theory and Applications of Parametric Weighted Finite Automata

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

vorgelegt von

German Tischler

aus
Würzburg

Würzburg, 2008

Eingereicht am: 1. Februar 2008
bei der Fakultät für Mathematik und Informatik
1. Gutachter: Prof. Dr. J. Albert
2. Gutachter: Prof. Dr. J. Kari
Tag der mündlichen Prüfung: 3. Juli 2008

Acknowledgments

I am grateful to Prof. Dr. Jürgen Albert for the support he gave me and for supervising this thesis, especially for the many fruitful discussions on weighted automata, without which the writing of this thesis would not have been possible in the form in which it has happened. Furthermore, I am grateful to Prof. Dr. Jarkko Kari for acting as a referee for this thesis.

I want to thank Alexander Eckl for proof-reading this thesis.

Finally I want to thank my mother and my beloved girlfriend Jutta Wellmann for their love, support and understanding.

I dedicate this thesis to my father.

Würzburg, January 2008 German Tischler

Contents

Chapter 1

Introduction

In computer science the term *automaton* (cf. [58]) is used for any kind of device that processes a given input and produces some kind of output. The study of these devices is called automata theory. In automata theory the input of an automaton is usually a word, i.e. a sequence of letters. Automata theory has many important applications in computer science and other areas. These applications include compilers for programming languages, parsers for natural languages, text searching, data mining, software engineering and many more. Automata can be classified by various criteria. One important criterion is how much memory an automaton is allowed to use and how it is allowed to access it. An automaton which independent of the input uses only a finite amount of memory is called a *finite automaton*, where the finite memory is usually represented as a finite set of states. Another criterion for the classification of automata is, how the automaton is allowed to read the input, e.g. is it allowed to read the input only sequentially and just once, or is it allowed to read the input an unlimited number of times and not necessarily sequentially. A third important criterion is, what kind of output the automaton produces and at what time it produces the output, i.e. is it allowed to produce some amount of output for each computation step or is it only allowed to produce output at the end of the computation. An automaton, which produces output for every computation step is called a transducer. One which only produces output at the end of the computation is called an acceptor or recognizer. Finite automata, recognizers as well as transducers, have been studied extensively. The two most well known finite transducer variants are called Moore and Mealy machines. These machines are, except for their behavior for the empty word (i.e. the word consisting of no letters), equally powerful. Finite recognizers can be classified by what kind of output they generate. The most simple variants, which we call Boolean finite recognizers, produce either the output *true* or *false*, i.e. they accept or reject the given input word. The two most well known models of Boolean finite recognizers are called *deterministic finite automata* (DFA) and *nondeterministic finite automata* (NFA). Both, DFA and NFA are so called *finite state machines*, i.e. the memory of the automaton is given as a finite non-empty set of n states, which the automaton can either assume or not assume. A DFA assumes at most one state after reading some input word, thus it has a memory that corresponds roughly to a register, which is able to hold one number with $\lceil \log_2 n \rceil$ binary digits. The number of states an NFA can possibly assume after

reading a word is ranging from zero to n, which corresponds to an amount of storage equaling one register, which is able to hold a number consisting of n binary digits. In both cases, for DFA and NFA, the states are partitioned into accepting and non-accepting states. In the DFA case exactly one state is marked as the initial state, which is the state the automaton assumes at the beginning of each computation. In the NFA case there can be multiple initial states. A DFA or NFA accepts a word, i.e. the output of the automaton is *true*, if and only if it assumes an accepting state at the end of the computation. DFA and NFA are equivalent in the sense that the acceptance behavior of an NFA can be simulated by a DFA and vice versa. If an NFA N is to be simulated by a DFA, then the smallest DFA simulating N can have a number of states, which is exponential in the number of states of N. The result of the computation of a DFA for a given word is however obtained more easily, as the DFA assumes at most one state at any time. Boolean finite recognizers are important in applications, as they correspond to the so called *regular sets* (cf. [58]). DFA can be used to determine in linear time, whether a given word is a member of a regular set. The term linear time denotes that the automaton uses no more than a fixed amount of time for any letter it processes, where this fixed amount of time does not depend on the given input word or the processed letter. The same is basically true for NFA, the time used to process one letter is however usually much higher in the case of an NFA. Regular sets are often given by *regular expressions*, which are very frequently used in compiler and parser construction. A regular expression is usually first translated to an NFA and the resulting NFA is translated to a DFA for evaluation. In some cases the step of transforming the NFA representation to an equivalent DFA representation is omitted, if the number of states required for a suitable DFA is too large to be handled by some employed real machine.

Weighted finite automata (WFA) are a generalization of NFA introduced by Schützenberger in [81]. An NFA assumes a subset of its states at any time, i.e. each state is either assigned the value *true* or *false*, depending on if the automaton currently assumes it. Instead of taking the values *true* or *false*, the states in a WFA can assume arbitrary elements of a given semiring and instead of producing either the output *true* or *false*, the output of a WFA can be an arbitrary element of some semiring also. Like NFA correspond to regular expressions, WFA correspond to rational formal power series (cf. [10]). WFA have a wide range of applications, including language and speech processing [75], pattern matching [46, 45, 62], image processing and transformation [27], computation of real functions [20, 35, 19], image compression [21, 26, 50, 65, 64, 30, 29, 63, 24], texture analysis [40] and fractal geometry [28]. Note that a weighted finite automaton is not necessarily a finite automaton in the sense that it only uses a finite amount of memory independent of the input. Depending on the semiring, the storage of a single semiring element may require an unbounded amount of space. Thus the automata are in general only finite in the sense that the number of concurrently required semiring element instances at each time is finite.

Parametric weighted finite automata (PWFA) are a generalization of WFA and were introduced by Albert and Kari in [4]. PWFA are different from WFA in two ways. The first is that PWFA consist of multiple WFA running synchronously, i.e. instead of computing one semiring element for each input word, a PWFA computes a finite sequence of semiring elements for each input word. The number of synchronously running WFA is called the dimension of the PWFA. This multi-dimensionality accounts for the attribute *parametric* in parametric weighted

finite automata. The second difference is that we do not consider, which output sequence a PWFA produces for which input word. We rather consider the set S of all sequences an observed PWFA produces infinitely often. If the underlying semiring is a topological space, then we consider all sequences s such that the automaton produces infinitely many sequences in every neighborhood of s as members of the set S also. PWFA are known to be more powerful than WFA. Some functions, which are not computable by WFA (cf. [36]), are computable by PWFA. This includes the real sine, cosine and exponential functions. PWFA can also be used to represent attractors of iterated function systems (IFS, cf. [8]), recurrent iterated function systems (RIFS, cf. [9]) and mutually recursive function systems (MRFS, cf. [17]). In contrast to real unary alphabet WFA, real unary alphabet PWFA are able to represent graphically interesting sets, e.g. the unit circle in the Euclidean plane.

In this thesis we will study the theory and applications of parametric weighted finite automata. We will introduce some required notations and definitions in chapter 2. Chapter 3 is dedicated to a short formal introduction to finite automata and weighted finite automata. In chapter 4 we will formally introduce parametric weighted finite automata, give a discussion of the PWFA definition and study some structural properties of PWFA. We will show that each PWFA computable set is computable by a PWFA that has at most two different input symbols. In chapter 5 we present some results concerning the set of sets computable by PWFA. We will show that this set is, assuming that the observed PWFA fulfill certain constraints, closed under the set union operation, affine transformation, iterated affine transformation and regular restriction. Concerning the set intersection and complement operations, we give some hints, why we conjecture that the respective closures do not hold. Furthermore, we will show that the membership, emptiness and equivalence problems for PWFA computed sets are in general recursively undecidable, implying that there is no effective state minimization algorithm for PWFA. We will discuss the relationship of PWFA to IFS, RIFS and MRFS in chapter 6. We will show that it is decidable, whether a complex square matrix, in which each element has a rational real and imaginary part, denotes a contraction mapping. Furthermore, we will give constructive proofs that each attractor of an affine IFS, RIFS or MRFS is PWFA computable. We will also show that the converse does not hold, i.e. there are PWFA computable sets, which cannot be generated by IFS, RIFS and MRFS. In chapter 7 we present various real and complex functions computable by PWFA. These include real and complex polynomials on certain compact sets, real dyadic compactly supported finite impulse response scaling functions and wavelets, the real sine, cosine and exponential functions, some real and complex rational functions and the complex exponential function. We will show that certain types of splines and spline patches are easily representable using PWFA in chapter 8. In particular, these are Bézier curves, Bézier patches, Catmull-Rom splines and B-splines. At the end of chapter 8 we will present some spline applications, which can be implemented using PWFA. In chapter 9 we will show how sets computed by PWFA of suitable dimensions can be interpreted as images, 3d scenes, sets of polygonal surfaces and movies. Furthermore, we will present some decoding algorithms for PWFA. Finally we summarize the given results and present some interesting open problems in chapter 10. The color plates and bibliography can be found at the end of the thesis. The bibliography contains some URLs (uniform resource locators) to resources found on the WWW

(world wide web). It is in the nature of the WWW that some degree of volatility adheres to these resources. Therefore we have noted for each URL, at which time we have checked that the corresponding resource was actually available in the form we have observed it.

Chapter 2

Notations and definitions

In this chapter we introduce some basic notations and definitions used in this thesis. Most of these are very common and we will thus only provide references to standard literature.

2.1 Referenced notations and definitions

We use the set notation and definitions of operations on sets as defined by Kuratowski in [66], where we denote the limes inferior and limes superior of a sequence of sets $(S_i)_{i=1}^{\infty}$ by $\liminf_{i\to\infty} S_i$ and $\limsup_{i\to\infty} S_i$ respectively. The terms Cartesian product, relation, function and closely related terms like image, preimage, etc. and terms in the area of topology we use are defined by Munkres in [76]. We call a function $f : A \mapsto B$ for some sets A and B total, if $f(a)$ is defined for each $a \in A$. Furthermore, we will denote each pair of a set X and a metric m on X by (X,m). Throughout this thesis we will assume that each topological space X we consider satisfies Hausdorff's separation axiom, i.e. each sequence in X has at most one limit in X. We denote the lower and upper limit of a sequence $(S_i)_{i=1}^{\infty}$ of subsets of a topological space as defined by Kuratowski in [66] by $\overline{\liminf}_{i\to\infty} S_i$ and $\overline{\limsup}_{i\to\infty} S_i$ respectively. The definitions we use in the area of linear algebra like vector space, inner product, etc. are given by Curtis in [31]. The definitions of the terms semiring, module, Hadamard product and related terms we use are the same as those given by Berstel and Reutenauer in [10]. We will call a semiring $\mathcal{S}$ metric or topological, if $\mathcal{S}$ is a metric or topological space respectively. Furthermore, we will (in lack of a short name for the elements of modules) call the elements of modules vectors, even if the underlying module is not a vector space. The notation and definitions of operations on finite words and finite languages we use can be found in [58] by Hopcroft and Ullman and those for infinite words in [85] by Staiger.

2.2 Interpretation of words as numbers

We will use the following definitions for the interpretation of words as numbers. They correspond to the representation of numbers in positional systems.

Definition 2.1 *We define $K[k]$ as the set $\{0,1,\ldots,k-1\}$ for each $k \in \mathbb{N}, k \geq 2$.*

Definition 2.2 *We define $\mathbb{Q}_U$ as the set $\mathbb{Q} \cap [0,1)$.*

Definition 2.3 *Let $k \in \mathbb{N}, k \geq 2$.*

1. *We define the function $g[k] : K[k]^* \mapsto \mathbb{N}$ by*

$$g[k](w) = \sum_{i=1}^{n} a_i k^{n-i} \tag{2.1}$$

for each $w = a_1 \dots a_n \in K[k]^*$.

2. *We define the function $q[k] : K[k]^* \mapsto \mathbb{Q}_U$ by*

$$q[k](w) = \sum_{i=1}^{n} a_i k^{-i} \tag{2.2}$$

for each $w = a_1 \dots a_n \in K[k]^*$.

3. *We define the function $r[k] : K[k]^{\omega} \mapsto [0,1]$ by*

$$r[k](w) = \sum_{i=1}^{\infty} a_i k^{-i} \tag{2.3}$$

for each $w = a_1 \dots \in K[k]^{\omega}$.

Definition 2.4 *We define the set $Q[k] \subset \mathbb{Q}_U$ as*

$$Q[k] = \bigcup_{w \in K_k^*} q[k](w) \tag{2.4}$$

for each $k \in \mathbb{N}, k \geq 2$. Furthermore, we define the function $\hat{q}[k] : K_k^* \mapsto Q[k]$ for each $k \in \mathbb{N}, k \geq 2$ by

$$\hat{q}[k](w) = q[k](w) \tag{2.5}$$

for each $w \in K_k^*$.

It is well known that the functions $g[k]$ and $r[k]$ are surjective and that the image generated by the function $q[k]$ for the set K_k^* is dense in $[0,1]$ for each natural number $k \geq 2$.

For each $k \in \mathbb{N}, k \geq 2$ we can assign a minimum element to each subset of $K[k]^*$ as given in the following definition.

Definition 2.5 *Let $k \in \mathbb{N}, k \geq 2$. We can impose a total order on $K[k]^*$ by defining that some element $v \in K[k]^*$ is smaller than some element $w \in K[k]^*$, if and only if either $|v| < |w|$ or $|v| = |w|$ and $q[k](v) < q[k](w)$. We call v the minimum element of $S \in \wp(K[k]^*)$, if and only if $v \in S$ and $v < w$ holds for all elements $w \in S, w \neq v$. Furthermore, we denote the minimum element of $S \in \wp(K[k]^*) \setminus \emptyset$ by $\min_q[k](S)$.*

Chapter 3

Weighted finite automata

3.1 Finite automata

An early definition of the term finite automaton can be found in [84]. The term *finite* in finite automata stems from the fact that each finite automaton has a finite number of states. A finite automaton is assumed to work in discrete steps. It processes its input word over some alphabet Σ sequentially, character by character. The simplest class of finite automata is the class of finite automata that map words over Σ^* to the set $\{0,1\}$, i.e. each given input word is either accepted or rejected.

3.1.1 Deterministic finite automata

Definition 3.1 *A deterministic finite automaton (DFA) is a quintuple*

$$D = (Q, \Sigma, \delta, s, F),$$

where

- $Q = \{1, 2, \ldots, n\}$ *for some* $n \in \mathbb{N}^+$ *is a finite non-empty set of states,*
- $\Sigma = \{1, 2, \ldots, l\}$ *for some* $l \in \mathbb{N}^+$ *is an input alphabet,*
- $\delta : Q \times \Sigma \mapsto Q$ *is the transition function,*
- $s \in Q$ *is the initial state and*
- $F \subset Q$ *is the set of final states.*

δ *is not necessarily a total function. The extended transition function* $\hat{\delta} : Q \times \Sigma^* \mapsto Q$ *is defined inductively by*

$$\hat{\delta}(q, \varepsilon) = q \tag{3.1}$$

for each $q \in Q$ *and*

$$\hat{\delta}(q, aw) = \begin{cases} \hat{\delta}(\delta(q,a), w) & \text{if } \delta \text{ is defined for } (q,a) \\ \text{undefined} & \text{otherwise} \end{cases} \tag{3.2}$$

for each $q \in Q$, $a \in \Sigma$ and $w \in \Sigma^*$. D computes the function $f : \Sigma^* \mapsto \{0,1\}$ defined by

$$f(w) = \begin{cases} 1 & \text{if } \hat{\delta} \text{ is defined for } (s,w) \text{ and } \hat{\delta}(s,w) \in F \\ 0 & \text{otherwise} \end{cases} \tag{3.3}$$

for each $w \in \Sigma^*$. The language $L(D)$ accepted by D is defined as

$$L(D) = \{w | w \in \Sigma^*, f(w) = 1\} \ . \tag{3.4}$$

Furthermore, we define the transition relation R of D as

$$R = \{(s,a,t) | s,t \in Q, a \in \Sigma, \delta(s,a) = t\} \ . \tag{3.5}$$

We will use subscript notations like Q_X and f_X in the following to denote the set of states Q or the function f computed by the automaton X.

We call a DFA D complete, if its transition function δ is a total function. Otherwise we call D incomplete. It is trivial to show that for every DFA D, there is an equivalent complete DFA D' accepting $L(D') = L(D)$. A complete DFA D is in exactly one state $q \in Q$ after reading a word $w \in \Sigma^*$ and this state is $q = \hat{\delta}(w)$. If q is a final state, then w is accepted by the automaton, otherwise it is rejected. If the extended transition function of an incomplete DFA D is not defined for some $w \in \Sigma^*$, then we may imagine that D does not assume any state after reading w (especially D does not assume a final state and thus does not accept w). Hence, a DFA can be in at most one state after reading a word $w \in \Sigma^*$.

3.1.2 Nondeterministic finite automata

A deterministic finite automaton assumes at most one state at any time. If it assumes a certain state and consumes an input symbol, there is at most one successor state it can assume, which is determined by the transition function. In contrast, if a nondeterministic finite automaton assumes a certain state and consumes a certain symbol, it may depending on the transition function be allowed to enter several successor states.

Definition 3.2 *A nondeterministic finite automaton (NFA) is a quintuple*

$$N = (Q, \Sigma, \delta, I, F),$$

where

- *$Q = \{1,2,\ldots,n\}$ for some $n \in \mathbb{N}^+$ is a finite non-empty set of states,*
- *$\Sigma = \{1,2,\ldots,l\}$ for some $l \in \mathbb{N}^+$ is an input alphabet,*
- *$\delta : Q \times \Sigma \mapsto \wp(Q)$ is the transition function,*
- *$I \subset Q$ is the set of initial states and*
- *$F \subset Q$ is the set of final states.*

We assume without loss of generality that δ is a total function (if δ is not defined for some argument pair $(q,a) \in Q \times \Sigma$, then we define $\delta(q,a) = \emptyset$). The extended transition function $\hat{\delta} : \wp(Q) \times \Sigma^* \mapsto \wp(Q)$ is defined inductively by

$$\hat{\delta}(S, \varepsilon) = S \tag{3.6}$$

for each $S \subset Q$ *and*

$$\hat{\delta}(S, wa) = \bigcup_{s \in \hat{\delta}(S,w)} \delta(s, a) \tag{3.7}$$

for each $S \subset Q$, $w \in \Sigma^*$ *and* $a \in \Sigma$. *N computes the function* $f : \Sigma^* \mapsto \{0,1\}$ *defined by*

$$f(w) = \begin{cases} 1 & \text{if } \hat{\delta}(I,w) \cap F \neq \emptyset \\ 0 & \text{otherwise} \end{cases} \tag{3.8}$$

for each word $w \in \Sigma^*$. *The language* $L(N)$ *accepted by N is defined as*

$$L(N) = \{w | w \in \Sigma^*, f(w) = 1\} \ . \tag{3.9}$$

3.1.3 Regular languages

It is well known that DFA and NFA are able to accept the same set of languages, which are called regular languages.

Definition 3.3 *The regular languages over the finite alphabet* Σ *are the smallest family of languages satisfying the following conditions 1–3.*

1. $L = \emptyset$, $L = \{\varepsilon\}$ *and* $L = \{a\}$ *for all* $a \in \Sigma$ *are regular languages.*
2. *If* L_1 *and* L_2 *are regular languages, then* $L_1 \cup L_2$ *and* $L_1 L_2$ *are regular languages.*
3. *If L is a regular language, then* L^* *is a regular language.*

Regular languages are closed under most of the common language operations, including union, intersection, complement, concatenation and the Kleene star. Each regular language can be written as a regular expression (cf. [58]). Regular languages are interesting in applications, because they can be accepted in linear time with finite memory.

There are effective algorithms to transform each regular language given in the form of an NFA, DFA or regular expression into an equivalent NFA, DFA or regular expression. For each regular language L there is a (upto bijective state renaming) unique state minimal DFA D accepting L, i.e. D accepts L and there is no DFA D' having a smaller number of states than D, which accepts L. This minimal automaton can be computed effectively.

3.1.4 Display of finite automata as augmented graphs

A nondeterministic finite automaton $A = (Q, \Sigma, \delta, I, F)$ can be displayed as an augmented graph with labeled edges and vertices. The states are shown as vertices. State $m \in Q$ is labeled by the state number m and the values of the characteristic functions χ_I and χ_F for m written as $I : \chi_I(m)$ and $F : \chi_F(m)$ respectively. The transition function is depicted by the directed edges of the graph. Let succ(i) denote the set of states that can be reached from state $i \in Q$ by one transition. We draw a directed edge from state $i \in Q$ to state $j \in Q$, if and only if succ(i) contains state j, where the edge is labeled by all symbols $a \in \Sigma$, for which $\delta(i,a)$ contains state j.

A deterministic finite automaton $A = (Q, \Sigma, \delta_D, s, F)$ is displayed in the same way as the trivially obtained nondeterministic finite automaton $B = (Q, \Sigma, \delta_N, \{s\},$

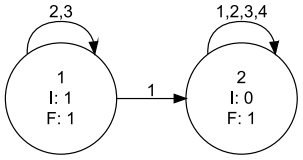


Figure 3.1: Graphical representation of a deterministic finite automaton accepting the regular language given by the regular expression $(((2+3)^*)(\varepsilon+(1(\Sigma^*))))$ over the alphabet $\Sigma = \{1,2,3,4\}$.

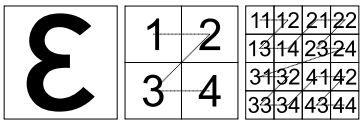


Figure 3.2: Interpretation of words over $\Sigma = \{1,2,3,4\}$ as image coordinates according to the so called Morton or Z-order for the word lengths 0, 1 and 2 (left to right).

F), where $\delta_N(i,a) = \{\delta_D(i,a)\}$, if $\delta_D(i,a)$ is defined and $\delta_N(i,a) = \emptyset$ otherwise for each $i \in Q$ and $a \in \Sigma$.

Example 3.1 *Let $\Sigma = \{1,2,3,4\}$. Furthermore, let the DFA D be given by*

$$D = (\{1,2\}, \Sigma, \delta_1, 1, \{1,2\}), \tag{3.10}$$

where $\delta_1(1,1) = 2$, $\delta_1(1,2) = \delta_1(1,3) = 1$, $\delta_1(2,a) = 2$ for each $a \in \Sigma$ and δ_1 undefined otherwise. D accepts the regular language denoted by the regular expression $(((2+3)^*)(\varepsilon+(1(\Sigma^*))))$ and is depicted in Figure 3.1. The equivalent NFA N defined by

$$N = (\{1,2\}, \Sigma, \delta_2, \{1\}, \{1,2\}), \tag{3.11}$$

where $\delta_2(1,1) = \{2\}$, $\delta_2(1,2) = \delta_2(1,3) = \{1\}$, $\delta_2(2,a) = \{2\}$ for each $a \in \Sigma$ and δ_2 empty otherwise, is depicted in the same way as D.

The set of words Σ^k can be interpreted as pixel coordinates of an image of size $2^k \times 2^k$ for each $k \in \mathbb{N}$. One such interpretation according to the Morton- or Z-order is shown in Figure 3.2 for the word lengths 0 to 2. The construction for higher word lengths is straight-forward. The automaton D can thus be interpreted as a $2^k \times 2^k$ bi-level image for each $k \in \mathbb{N}$. A pixel is assigned the color white, if the automaton accepts the word corresponding to the pixel position. Otherwise it is assigned the color black. The images obtained for the word lengths 0 to 5 are shown in Figure 3.3.

3.2 Weighted finite automata

Weighted finite automata (WFA) are a generalization of NFA. They were introduced by Schützenberger in [81]. The output of a WFA can be an element of any semiring instead of an element of the Boolean semiring. WFA have a wide range

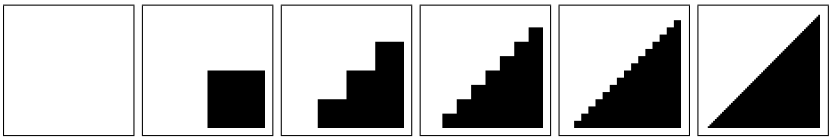


Figure 3.3: Images produced by the automaton shown in Figure 3.1 for the word lengths 0 to 5 (scaled to the same size).

of applications, including language and speech processing [75], pattern matching [46,45,62], image processing and transformation [27], computation of real functions [20,35,19], image compression [21,26,50,64,65,30,29,63,24], texture analysis [40] and fractal geometry [28].

3.2.1 Definition

Definition 3.4 *A weighted finite automaton (WFA) over a semiring $\mathcal{S}$ is a quintuple (Q,Σ,W,I,F), where*

- *$Q = \{1,2,\ldots n\}$ for some $n \in \mathbb{N}^+$ is a finite non-empty set of states,*
- *$\Sigma = \{0,1,\ldots,l-1\}$ for some $l \in \mathbb{N}^+$ is an input alphabet,*
- *$W = (A_0,A_1,\ldots,A_{l-1})$ is a sequence of total transition functions $A_i : Q \times Q \mapsto \mathcal{S}$ for $i = 0,1,\ldots,l-1$,*
- *$I : Q \mapsto \mathcal{S}$ is a total function called the initial distribution and*
- *$F : Q \mapsto \mathcal{S}$ is a total function called the final distribution.*

As n is finite, each transition function A_k for $k = 0,\ldots,l-1$ can also be interpreted as a matrix $A_k \in \mathcal{S}^{n\times n}$, the initial distribution I can be interpreted as a row vector $I \in \mathcal{S}^{1\times n}$ and the final distribution F as a column vector $F \in \mathcal{S}^{n\times 1}$. The function $f : \Sigma^* \mapsto \mathcal{S}$ computed by the automaton is defined by

$$f(w) = \underbrace{\sum_{p=(q_0,q_1,\ldots,q_m)\in Q^{m+1}}}_{\text{All possible paths of length } m} \underbrace{\left(I(q_0)\left(\prod_{i=1}^{m} A_{a_i}(q_{i-1},q_i)\right)F(q_m)\right)}_{\text{Value computed for path } p} \tag{3.12}$$

for each $w = a_1\ldots a_m \in \Sigma^*$. Let the morphism $\mu(w) : \Sigma^* \mapsto \mathcal{S}^{n\times n}$ be defined by $\mu(w) = \prod_{i=1}^{m} A_{a_i}$ for each word $w = a_1\ldots a_m \in \Sigma^*$. Then f can be written in matrix and vector form as

$$f(w) = I\mu(w)F \quad . \tag{3.13}$$

Furthermore, we define the transition relation R of the WFA as the set

$$R = \{(s,a,t,w) | s,t \in Q, a \in \Sigma, w \in \mathcal{S}, w \neq 0, A_a(s,t) = w\} \quad . \tag{3.14}$$

If $\mathcal{S}$ is a topological semiring, then we define the ω function f^ω computed by the automaton on infinite words by

$$f^\omega(w) = \lim_{i\to\infty} f(a_1\ldots a_i) \tag{3.15}$$

for each $w = a_1 a_2 \ldots \in \Sigma^\omega$. If this limit does not exist for some $w \in \Sigma^\omega$, then we say that f^ω is undefined for w.

Remark 3.1 *In contrast to the other definitions of automata provided in this thesis, we choose the first symbol of the alphabet Σ of a weighted finite automaton as 0 instead of 1, because the input words are often interpreted as rational or real numbers using the functions $q[k]$ and $r[k]$ for $k \in \mathbb{N}, k \geq 2$.*

A similar WFA definition is given by Berstel and Reutenauer in [10], where instead of the term weighted finite automaton the notion of a *linear representation* of a recognizable formal power series (cf. [10]) is used.

Let $B = (Q = \{1, \ldots, n\}, \Sigma, W, I, F)$ denote a WFA over some semiring $\mathcal{S}$. Furthermore, let f_i for $i = 1, \ldots, n$ denote the function computed by the WFA $B_i = (Q, \Sigma, W, I_i, F)$ that equals B except for the initial distribution I_i, which is set to the i'th unit vector. We call f_i the state function of state i in B. A closer look at equation 3.13 yields that

$$\begin{aligned} f_i(av) &= I_i\mu(av)F = I_i(\mu(a)\mu(v))F = (I_i\mu(a))\mu(v)F \\ &= \textstyle\sum_{j=1}^{n} (A_a(i,j)I_j)\mu(v)F = \sum_{j=1}^{n} A_a(i,j) f_j(v) \end{aligned} \tag{3.16}$$

holds for each $a \in \Sigma, v \in \Sigma^*$. This means that the state function of state i is determined by $|\Sigma|$ linear combinations of other state functions. The function

$$f(w) = \sum_{j=1}^{n} I(j) f_j(w) \tag{3.17}$$

computed by B for each $w \in \Sigma^*$ is a linear combination of its state functions.

3.2.2 Examples

We give three WFA examples.

Example 3.2 *The WFA*

$$X = (\{1\}, \Sigma, ((1), (1), \ldots, (1)), (v), (1))$$

over an arbitrary semiring $\mathcal{S}$ computes the constant function $f \equiv v \in \mathcal{S}$ for each input alphabet Σ. If $\mathcal{S}$ is a topological semiring, then X also computes the ω function $f^\omega : \Sigma^\omega \mapsto \mathcal{S}$ defined by $f^\omega(w) = v$ for each word $w \in \Sigma^\omega$.

Example 3.3 *The WFA*

$$X = (\{1,2\}, \Sigma = \{0, 1, \ldots, k-1\}, W = (A_0, A_1, \ldots, A_{k-1}), (0 \;\; 1), (1 \;\; 0)^T)$$

over the complex numbers computes the function $f(w) = q[k](w)$ (i.e. the rational number $0.a_1 \ldots a_n$ to the base k) for each word $w = a_1 \ldots a_n \in \Sigma^*$ and $k \geq 2$, if we define

$$A_a = \begin{pmatrix} 1 & 0 \\ \frac{a}{k} & \frac{1}{k} \end{pmatrix} \tag{3.18}$$

for each $a \in \Sigma$. After reading a word of length m, state 2 contains the value $\frac{1}{k}^m$. The result of the function is accumulated in state 1, which holds the value $q[k](a_1 \ldots a_m)$ after reading the prefix $a_1 \ldots a_m$ of the word $w = a_1 \ldots a_n \in \Sigma^m \Sigma^*$. As X computes $q[k](w)$ for every finite word $w \in \Sigma^*$, it also computes the ω function $f^\omega : \Sigma^\omega \mapsto \mathbb{C}$ defined by $f^\omega(w) = r[k](w)$ for each $w \in \Sigma^\omega$.

Example 3.4 *The* ω *function defined by the WFA*

$$X = (\{1\},\{0,1\},((2),(2)),(1),(1))$$

over the real numbers is undefined for each infinite input word.

3.2.3 Functions over semirings

We may understand WFA as devices computing functions in the way described in the following definitions.

Definition 3.5

1. *Let* D *denote a finite or countable set and let* $X = (Q,\Sigma = \{0,1,\ldots,k-1\}, W,I,F)$ *be a WFA over the semiring* $\mathcal{S}$ *computing the function* $f : \Sigma^* \mapsto \mathcal{S}$*, where* $k \geq 2$*. Furthermore, let* $\mathcal{M} : \Sigma^* \mapsto D$ *denote a surjective total function. We call such a function a word interpretation function. Let the function* $\mathcal{M}^{-1} : D \mapsto \wp(\Sigma^*)$ *be defined by* $\mathcal{M}^{-1}(d) = \{w | w \in \Sigma^*, \mathcal{M}(w) = d\}$ *for each* $d \in D$*. Furthermore, let the function* $f' : \wp(\Sigma^*) \mapsto \mathcal{S}$ *be defined by*

$$f'(G) = \begin{cases} f(\min_q[k](G)) & \text{if } G \neq \emptyset \text{ and there exists some } d \in \mathcal{S} \text{ such that} \\ & f(g) = d \text{ for all } g \in G \\ \text{undefined} & \text{otherwise} \end{cases} \tag{3.19}$$

for each $G \in \wp(\Sigma^*)$*. Then we define the function* $f_{\mathcal{M}} : D \mapsto \mathcal{S}$ *by*

$$f_{\mathcal{M}}(d) = f'(\mathcal{M}^{-1}(d)) \tag{3.20}$$

for each $d \in D$ *and say that* X *computes* $f_{\mathcal{M}}$ *under* $\mathcal{M}$*.*

2. *Let* D *be a set having at most the cardinality of the real numbers and let* $X = (Q,\Sigma,W,I,F)$ *be a WFA over the topological semiring* $\mathcal{S}$ *computing the* ω *function* $f^\omega : \Sigma^\omega \mapsto \mathcal{S}$*, where* $k \geq 2$*. Furthermore, let* $\mathcal{M}^\omega : \Sigma^\omega \mapsto D$ *denote a surjective total function. We call such a function an* ω *word interpretation function. Let* $u : \wp(\mathcal{S}) \mapsto \mathcal{S}$ *be defined by*

$$u(S) = \begin{cases} \text{the single element of } S & \text{if } |S| = 1 \\ \text{undefined} & \text{otherwise} \end{cases} \tag{3.21}$$

for each $S \in \wp(\mathcal{S})$ *and let* $f^{\omega^S}(G) : \wp(\Sigma^\omega) \mapsto \wp(\mathcal{S})$ *be defined by*

$$f^{\omega^S}(G) = \bigcup_{g \in G} \{f^\omega(g)\} \tag{3.22}$$

for each $G \in \wp(\Sigma^\omega)$*. Let the function* $\mathcal{M}^{\omega^{-1}} : D \mapsto \wp(\Sigma^\omega)$ *be defined by* $\mathcal{M}^{\omega^{-1}}(d) = \{w | w \in \Sigma^\omega, \mathcal{M}(w) = d\}$ *for each* $d \in D$*. Furthermore, let the function* $f^{\omega'} : \wp(\Sigma^\omega) \mapsto \mathcal{S}$ *be defined by*

$$f^{\omega'}(G) = \begin{cases} u(f^{\omega^S}(G)) & \text{if } G \neq \emptyset \text{ and there exists some } d \in \mathcal{S} \\ & \text{such that } f^\omega(g) = d \text{ for all } g \in G \\ \text{undefined} & \text{otherwise} \end{cases} \tag{3.23}$$

for each $G \in \wp(\Sigma^\omega)$. Then we define the function $f^\omega_{\mathcal{M}} : D \mapsto \mathcal{S}$ by

$$f^\omega_{\mathcal{M}}(d) = f^{\omega'}(\mathcal{M}^{\omega^{-1}}(d)) \tag{3.24}$$

for each $d \in D$ and say that X computes $f^\omega_{\mathcal{M}}$ under $\mathcal{M}^\omega$.

Definition 3.5 could be written in a much simpler way, if we would require the functions $\mathcal{M}$ and $\mathcal{M}^\omega$ to be bijective. However, some popular (ω) word interpretation functions, e.g. the functions $r[k]$ for $k \geq 2$, are only surjective and not bijective.

Definition 3.6

1. *Let D be a set, $\mathcal{S}$ a semiring, Σ an alphabet and $\mathcal{M} : \Sigma^* \mapsto D$ a word interpretation function. We say that the function $f : D \mapsto \mathcal{S}$ is WFA computable under $\mathcal{M}$, if there is some WFA X over $\mathcal{S}$ such that the function $f_{X_{\mathcal{M}}}$ computed by X under $\mathcal{M}$ is identical to f, i.e. $f(d) = f_{X_{\mathcal{M}}}(d)$ holds for each $d \in D$.*
2. *Let D be a set, $\mathcal{S}$ a topological semiring, Σ an alphabet and $\mathcal{M}^\omega : \Sigma^\omega \mapsto D$ an ω word interpretation function. We say that the function $f : D \mapsto \mathcal{S}$ is ω WFA computable under $\mathcal{M}^\omega$, if there is some WFA X over $\mathcal{S}$ such that the function $f^\omega_{X_{\mathcal{M}}}$ computed by X under $\mathcal{M}^\omega$ is identical to f, i.e. $f(d) = f^\omega_{X_{\mathcal{M}}}(d)$ holds for each $d \in D$.*

3.2.4 Closure under sum and product

Let $\mathcal{S}$ be a semiring and Σ an alphabet. Furthermore, let $f_1 : \Sigma^* \mapsto \mathcal{S}$ and $f_2 : \Sigma^* \mapsto \mathcal{S}$ be WFA computable functions. Then it is trivial to see that the function $(f_1 + f_2) : \Sigma^* \mapsto \mathcal{S}$ defined by $(f_1 + f_2)(w) = f_1(w) + f_2(w)$ for each $w \in \Sigma^*$ is also WFA computable. If $\mathcal{S}$ is commutative, then the function $(f_1 f_2) : \Sigma^* \mapsto \mathcal{S}$ defined by $(f_1 f_2)(w) = f_1(w) f_2(w)$ for each $w \in \Sigma^*$ is also WFA computable, which we show in the next lemma. The same result expressed in terms of formal power series can be found as Theorem 5.3 in [10]. In the proof of the lemma the product automaton is constructed as the tensor product of the input automata. We will use alphabets starting at 1 instead of 0 in the proof to simplify notations.

Lemma 3.1 *Let X and Y be WFA over the commutative semiring $\mathcal{S}$ such that $\Sigma_X = \Sigma_Y$. There is a WFA Z over the semiring $\mathcal{S}$ such that $\Sigma_Z = \Sigma_X$ and Z computes the function $f_Z(w) = f_X(w) f_Y(w)$ for each $w \in \Sigma^*$.*

Proof: Assume that $Q_X = \{1, \ldots, n_X\}$ for some $n_X \in \mathbb{N}^+$ and $Q_Y = \{1, \ldots, n_Y\}$ for some $n_Y \in \mathbb{N}^+$. Furthermore, let $\Sigma = \Sigma_X$. We choose Q_Z as the set $\{(1,1), \ldots (1, n_Y), (2,1), \ldots, (n_X, n_Y)\}$. Q_Z can be linearized by mapping each pair (o, p) for $1 \leq o \leq n_X, 1 \leq p \leq n_Y$ to $(o-1)n_Y + p$. Thus Z has one state for each pair (o, p) such that $o \in Q_X$ and $p \in Q_Y$. We want

$$I_Z \mu_Z(w)((o, p)) = (I_X \mu_X(w))(o)(I_Y \mu_Y(w))(p) \tag{3.25}$$

to hold for each $w \in \Sigma^*$, $o \in Q_X$ and $p \in Q_Y$. Setting $I_Z((o, p)) = I_X(o) I_Y(p), o \in Q_X, p \in Q_Y$ is apparently a valid choice to satisfy the case $w = \varepsilon$. We define the transition matrix A_{Z_b} as

$$A_{Z_b}((k, l), (o, p)) = A_{X_b}(k, o) A_{Y_b}(l, p) \tag{3.26}$$

for each $k,o \in Q_X$, $l,p \in Q_Y$ and $b \in \Sigma$. Let $\hat{f}_T(u)$ be defined by

$$\hat{f}_T(u) = I_T \mu_T(u) \tag{3.27}$$

for each WFA T over the alphabet Σ and all $u \in \Sigma^*$. Now assume that $w = va$ for $v \in \Sigma^*$ and $a \in \Sigma$. Then we observe that

$$\begin{aligned}
\hat{f}_Z(w)((o,p)) &= \sum_{k=1}^{n_X} \sum_{l=1}^{n_Y} \left(\hat{f}_Z(v)((k,l))\right) \left(A_{Z_a}((k,l),(o,p))\right) \\
&= \sum_{k=1}^{n_X} \sum_{l=1}^{n_Y} \left(\hat{f}_X(v)(k)\hat{f}_Y(v)(l)\right) \left(A_{X_a}(k,o)A_{Y_a}(l,p)\right) \\
&= \sum_{k=1}^{n_X} \sum_{l=1}^{n_Y} \left(\hat{f}_X(v)(k)A_{X_a}(k,o)\right) \left(\hat{f}_Y(v)(l)A_{Y_a}(l,p)\right) \\
&= \sum_{k=1}^{n_X} \left(\hat{f}_X(v)(k)A_{X_a}(k,o)\right) \sum_{l=1}^{n_Y} \left(\hat{f}_Y(v)(l)A_{Y_a}(l,p)\right) \\
&= \sum_{k=1}^{n_X} \left(\hat{f}_X(v)(k)A_{X_a}(k,o)\right) \hat{f}_Y(w)(p) \\
&= \hat{f}_Y(w)(p) \sum_{k=1}^{n_X} \left(\hat{f}_X(v)(k)A_{X_a}(k,o)\right) \\
&= \hat{f}_Y(w)(p)\hat{f}_X(w)(o) \\
&= \hat{f}_X(w)(o)\hat{f}_Y(w)(p) \quad .
\end{aligned} \tag{3.28}$$

Note that we have commuted the factors of some products in equation 3.28, so the constraint that the semiring is commutative is required in this proof. Furthermore, setting $F_Z((o,p)) = F_X(o)F_Y(p)$ for all $o \in Q_X, p \in Q_Y$ yields

$$\begin{aligned}
f_Z(w) &= I_Z \mu_Z(w) F_Z \\
&= \hat{f}_Z(w) F_Z \\
&= \sum_{k=1}^{n_X} \sum_{l=1}^{n_Y} \hat{f}_Z(w)((k,l))F_Z((k,l)) \\
&= \sum_{k=1}^{n_X} \sum_{l=1}^{n_Y} \hat{f}_X(w)(k)\hat{f}_Y(w)(l)F_X(k)F_Y(l) \\
&= \sum_{k=1}^{n_X} \sum_{l=1}^{n_Y} \left(\hat{f}_X(w)(k)F_X(k)\right) \left(\hat{f}_Y(w)(l)F_Y(l)\right) \\
&= \sum_{k=1}^{n_X} \left(\hat{f}_X(w)(k)F_X(k)\right) \sum_{l=1}^{n_Y} \left(\hat{f}_Y(w)(l)F_Y(l)\right) \\
&= \sum_{k=1}^{n_X} \left(\hat{f}_X(w)(k)F_X(k)\right) f_Y(w) \\
&= f_Y(w) \sum_{k=1}^{n_X} \left(\hat{f}_X(w)(k)F_X(k)\right) \\
&= f_Y(w) f_X(w) \\
&= f_X(w) f_Y(w)
\end{aligned} \tag{3.29}$$

for each $w \in \Sigma^*$. □

The generalization of Lemma 3.1 to WFA over topological semirings and words of infinite length is only straightforward, if the ω functions computed by the automata X and Y are total. If for instance the automaton X computes an ω function that is undefined for each infinite word (cf. Example 3.4) and Y computes the constant zero function for each word, then the product automaton constructed in the proof computes the constant zero function, although it should compute a function that is undefined for every infinite word.

3.2.5 Polynomials over commutative semirings

Using the results given above, we show that WFA can be used to compute polynomials over commutative semirings.

Theorem 3.1 *Let Σ be an alphabet, $\mathcal{S}$ a commutative semiring and $D \subset \mathcal{S}$. If there exists a WFA computable word interpretation function $\mathcal{M} : \Sigma^* \mapsto D$, then every polynomial $p : D \mapsto \mathcal{S}$ defined by $p(x) = \sum_{i=0}^{d} c_i x^i$ for a natural number d and $c_i \in \mathcal{S}$ for $0 \leq i \leq d$ is WFA computable under $\mathcal{M}$.*

Proof: Assume that there exists a WFA X computing a word interpretation function $\mathcal{M} : \Sigma^* \mapsto D$. We proof the theorem by induction over the degree d of the polynomial p.

d=0: We have shown in Example 3.2 that each constant function is WFA computable.

d>0: Assume that each polynomial of depth smaller than d is WFA computable under $\mathcal{M}$. The polynomial p can be decomposed as $p(x) = c_0 + p'(x)x$, where p' is some polynomial of degree $d-1$. Let P' denote a WFA computing p' under $\mathcal{M}$. Then we obtain the WFA P computing p under $\mathcal{M}$ by first constructing a WFA P'' computing the product of the functions computed by P' and X and consecutively constructing P as a WFA computing the sum of the function computed by P'' and the constant function c_0.

□

Hence, we can for instance construct WFA computing each given complex polynomial on the set $D = Q[2] \subset \mathbb{C}$ by using the word interpretation function $\mathcal{M} = \hat{q}[2]$.

If A and B are WFA over some semiring $\mathcal{S}$ such that $\Sigma_A = \Sigma_B$ and A and B compute the functions f_A and f_B respectively, then $f_A + f_B$ can be computed by an automaton that has $|Q_A| + |Q_B|$ states and $f_A f_B$ can be computed by an automaton that has $|Q_A||Q_B|$ states. Let Σ be an alphabet, D a subset of some semiring $\mathcal{S}$, X a WFA computing a word interpretation function $f_X : \Sigma^* \mapsto D$ and $p : D \mapsto \mathcal{S}$ be a polynomial of degree d. Assume that $|Q_X| = q$ for some $q \in \mathbb{N}, q \geq 2$. Then the construction above yields an automaton P computing p under $\mathcal{M}$ such that P has

$$1 + (q+1) + (q^2+q+1) + \ldots + (q^d + \ldots + 1) = \sum_{i=0}^{d} \frac{q^{i+1}-1}{q-1} \tag{3.30}$$

states, if X is used to represent $\mathcal{M}$.

3.2.6 Functions computed under $r[k]$

WFA computing real ω functions under the ω word interpretation function $r[2]$ have been studied extensively (cf. [20, 34, 36]). It was shown that each real polynomial of degree d over the interval $[0,1]$ can be computed by a $d+1$ state real WFA under $r[2]$ (cf. [20]) and that polynomials are the only smooth functions computable by real WFA under $r[2]$ (cf. [36]). The generalization of the first result to complex polynomials computed over $[0,1]$ under $r[2]$ is straightforward, we just need to use a suitable initial distribution. We show in the next lemma that the generalization to $r[k]$ for $k \geq 2$ is also possible.

Lemma 3.2 *Let $p = \sum_{i=0}^{d} c_i x^i$ be a complex polynomial of degree d, where $c_i \in \mathbb{C}$ for $i = 0, \ldots, d$ and let $k \geq 2$. There is a $d+1$ state complex ω WFA over $\mathbb{C}$ computing p on $D = [0,1]$ under $r[k]$.*

Proof: We use the alphabet $\Sigma = \{0, 1, \ldots, k-1\}$ and construct a WFA P such that $f_P(w) = p(q[k](w))$ for each $w \in \Sigma^*$. If we construct P in such fashion, then clearly P also computes the ω function $f_P^{\omega}(w) = p(r[k](w))$ for each $w \in \Sigma^{\omega}$, as for each

ω word $w = a_1 a_2 \ldots \in \Sigma^\omega$ the limit $\lim_{i\to\infty} q[k](a_1 \ldots a_i)$ exists and polynomials are continuous. Let the function $x^m : \Sigma^* \mapsto \mathbb{C}$ be defined by $x^m(w) = (r[k](w))^m$ for each $m \in \mathbb{N}$. We firstly construct a WFA computing the polynomial x^d. A WFA computing an arbitrary constant function was given in Example 3.2. Now assume that m is a positive natural number and we have already constructed an automaton P with m states, where the state function f_{P_i} equals x^{i-1} for $1 \leq i \leq m$. Then the function x^m can be implemented by adding a single state $m+1$ to P such that $f_{m+1} = x^m$. If $w = \varepsilon$, then we have $x^m(w) = 0^m = 0$, implying that the final weight for state $m+1$ equals 0. If $w = a_1 a_2 \ldots a_j \in \Sigma^*$ such that $w \neq \varepsilon$, then

$$\begin{aligned} x^m(w) &= (0.a_1 \ldots a_j)^m \\ &= (0.a_1 + 0.0a_2 \ldots a_j)^m \\ &= (\tfrac{1}{k}a_1 + \tfrac{1}{k}0.a_2 \ldots a_j)^m \\ &= \tfrac{1}{k^m}(a_1 + 0.a_2 \ldots a_j)^m \\ &= \tfrac{1}{k^m}\textstyle\sum_{i=0}^{m} \binom{m}{i} {a_1}^i (0.a_2 \ldots a_j)^{m-i} \quad . \end{aligned} \tag{3.31}$$

Let $e_{t,i,a} \in \mathbb{R}$ be defined by

$$e_{t,i,a} = \frac{a^{t-i}\binom{t}{i}}{k^t} \tag{3.32}$$

for each $t \in \mathbb{N}$, $0 \leq i \leq t$ and $a \in \Sigma$. Then the new state $m+1$ computes the state function x^m, if we add the edges $(m+1, a, i+1, e_{m,i,a})$ for all $a \in \Sigma$ and $i = 0, \ldots, m$ to the transition relation of P.
P is now a $d+1$ state WFA such that the state function of state i for $1 \leq i \leq d+1$ is $f_i = x^{i-1}$. As the function computed by a WFA is a linear combination of its state functions, we can make P compute p by setting component $i+1$ of I_P to c_i for $i = 0, \ldots, d$. □

Apart from smooth functions, WFA can also be used to produce exact representations of some non-smooth functions like scaling functions and wavelets [19] under $r[2]$.

The use of the ω word interpretation function $r[2]$ is by far the most common choice in publications on WFA computing real functions. We will thus in the following use the term *a WFA computing a real function* as an abbreviation of the term *a WFA computing a real function under* $r[2]$.

Chapter 4

Parametric weighted finite automata

4.1 Definition

Parametric weighted finite automata (PWFA) are a multidimensional variant of weighted finite automata. In this thesis, we will only consider such PWFA, whose multidimensionality stems from the usage of multiple initial distributions instead of only one. We call the number of initial distributions used the dimension of the automaton. Other possibilities to define PWFA would be the introduction of either only multiple final distributions or both multiple initial and multiple final distributions. We will see that if we just consider the function a PWFA computes for single words, then its behavior can be equivalently stated as that of a WFA over some other semiring. The set of vectors computed by a PWFA can be defined in various ways. We have chosen the variant that was also used when PWFA were introduced by Albert and Kari in [4]. We will give a short enumeration of possible alternative interpretations in section 4.3.

Definition 4.1 *A parametric WFA (PWFA) of dimension $d \in \mathbb{N}^+$ over a semiring $\mathcal{S}$ is a quintuple $X = (Q, \Sigma, W, I, F)$, where*

1. *$Q = \{1, 2, \ldots, n\}$ for some $n \in \mathbb{N}^+$ is a finite non-empty set of states,*
2. *$\Sigma = \{1, \ldots, l\}$ for some $l \in \mathbb{N}^+$ is a finite alphabet,*
3. *$W = (A_1, \ldots, A_l), A_i \in \mathcal{S}^{n \times n}$ are the transition matrices,*
4. *$I = (I_1, I_2, \ldots, I_d), I_j \in \mathcal{S}^n$ are the initial distributions, where the I_j are the rows of the matrix I and*
5. *$F \in \mathcal{S}^n$ is the final distribution.*

We will use the topological closure operator in some of the following equations. If $\mathcal{S}$ is not a topological semiring, then we substitute this operator by the corresponding identity map, i.e. $\overline{A} = A$ for each $A \subset \mathcal{S}$.

The word function or function $f : \Sigma^* \mapsto \mathcal{S}^d$ computed by the automaton is defined as

$$f(w) = IA_{a_1}A_{a_2}\ldots A_{a_i}F = I\prod_{j=1}^{i} A_{a_j}F = I\mu(w)F \tag{4.1}$$

for each $w = a_1 a_2 \dots a_i \in \Sigma^*$ *and the computed set* $S(X)$ *is*

$$S(X) = \overline{\limsup_{j \to \infty}} S_j(X) \tag{4.2}$$

where

$$S_j(X) = \left\{ f(v) | v \in \Sigma^j \right\} \tag{4.3}$$

for each $j \in \mathbb{N}$. *Furthermore, we define the sequence of sets* $(T_j(X))_{j=0}^{\infty}$ *by defining*

$$T_j(X) = \overline{S_{\geq j}(X)} = \overline{\bigcup_{k=j}^{\infty} S_k(X)} \tag{4.4}$$

for each $j \in \mathbb{N}$. *We will use*

$$T(X) = T_0(X) \tag{4.5}$$

as an abbreviation. The transition relation R *of the PWFA is defined as*

$$R = \{(s,a,t,w) | s,t \in Q, a \in \Sigma, w \in \mathcal{S}, w \neq 0, A_a(s,t) = w\} \ . \tag{4.6}$$

We call a state $s \in Q$ *an initial state for dimension* i, *if and only if* $1 \leq i \leq d$ *and* $I(i,s) \neq 0$. *Furthermore, we call a state* $s \in Q$ *an initial state, if and only if there exists some* i *for* $1 \leq i \leq d$ *such that* s *is an initial state for dimension* i. *We call a state* $s \in Q$ *a final state, if and only if* $F(s) \neq 0$.

We denote the set of all sets computable by PWFA of dimension d *over the semiring* $\mathcal{S}$ *by* $\mathcal{D}(\mathcal{S},d)$ *and define the set* $\mathcal{D}(\mathcal{S})$ *by*

$$\mathcal{D}(\mathcal{S}) = \bigcup_{d=1}^{\infty} \mathcal{D}(\mathcal{S},d) \tag{4.7}$$

for each semiring $\mathcal{S}$.

Remark 4.1 *We have defined the overline operator as the identity in the definition above for the case that the semiring* $\mathcal{S}$ *is not topological. This equals endowing* $\mathcal{S}$ *with the discrete topology and thus making it topological. As we can do this for every semiring* $\mathcal{S}$, *we will in the following without loss of generality assume that each semiring we consider is topological.*

4.2 Interpretation of the PWFA definition

The set of vectors $S(X)$ computed by a PWFA $X = (Q, \Sigma, W, I, F)$ over some semiring $\mathcal{S}$ is defined in equation 4.2. If $\mathcal{S}$ is endowed with the discrete topology, a vector v is a member of $S(X)$, if and only if there is some sequence of words of increasing length $w_1, w_2, \dots$ in Σ^* such that $f_X(w_j) = v$ for each $j = 1, 2, \dots$. If $\mathcal{S}$ is a semiring endowed with an arbitrary topology, then $v \in S(X)$ holds, if and only if there is some sequence of words of increasing length $w_1, w_2, \dots$ in Σ^* such that $\lim_{j \to \infty} f_X(w_j) = v$. In particular the set of word lengths $\mathbb{N} \setminus \{|w_1|, |w_2|, \dots\}$ can be infinite and w_i does not have to be a prefix of w_{i+1} for each $i \in \mathbb{N}^+$, as it is required in some ω WFA definitions. This is likely the main reason for the fact that some basic decision problems for PWFA are recursively undecidable, which we show in section 5.2.

In contrast to the set $S(X)$ that reflects the behavior the automaton shows infinitely often, the set $T(X)$ describes the behavior the automaton shows at least once. The equation $S(X) \subset T(X)$ holds for every PWFA X by definition. $S(X) = T(X)$ holds for most PWFA X used in applications like e.g. function drawing, spline display etc.

If $S(X) \neq T(X)$, we can effectively construct a PWFA Y for which $S(Y) = T(X)$.

Lemma 4.1 *Let X be a PWFA of dimension d over the semiring $\mathcal{S}$. There is a PWFA Y of dimension d over $\mathcal{S}$ computing $S(Y) = T(X)$.*

Proof: Assume that $Q_X = \{1, \ldots, n\}$ for some $n \in \mathbb{N}^+$. Y is obtained from X by adding an alphabet symbol $\zeta \notin \Sigma$ and assigning the $n \times n$ identity matrix to ζ. □

In contrast, it is in general impossible to construct a PWFA X' for a given PWFA X such that $T(X') = S(X)$, as $S(X)$ can be empty, but the T set of a PWFA is never empty.

We use the function h^* provided in the following definition for the comparison of arbitrary (i.e. not necessarily compact) subsets of a metric space.

Definition 4.2 *Let (X,m) be a metric space. Furthermore, let $\mathcal{P} = \wp(X)$ denote the power set of X. Then we define*

$$h^*(m)(A,B) = \begin{cases} 0 & \text{if } A = B = \emptyset \\ +\infty & \text{if } A \neq B = \emptyset \text{ or } B \neq A = \emptyset \\ \max\{\sup_{a\in A} \inf_{b\in B} m(a,b), & \text{otherwise} \\ \quad \sup_{b\in B} \inf_{a\in A} m(a,b)\} & \end{cases} \tag{4.8}$$

as a function $h^*(m) : \mathcal{P} \times \mathcal{P} \mapsto \overline{\mathbb{R}}_0^+$ for arbitrary sets $A, B \subset X$, where $\overline{\mathbb{R}} = \mathbb{R} \cup \{\pm\infty\}$ denotes the set of the so-called affinely extended real numbers.

The function h^* is in general not a metric, as it may produce the value $+\infty$ and thus is not a real function. It is however in some cases useful to determine the similarity of non-compact sets.

A notion that is important for transformations of PWFA computable sets is that of the ST-consistency of a PWFA.

Definition 4.3 *Let $(\mathcal{S}^d, m)$ be a metric space for some $d \in \mathbb{N}^+$, where $\mathcal{S}$ is a semiring. Furthermore, let X be a PWFA of dimension d over $\mathcal{S}$. X is called ST-*consistent under *m (or just ST-*consistent*, if the choice of m is obvious), if*

$$\lim_{i\to\infty} h^*(m)(S(X), T_i(X)) = 0. \tag{4.9}$$

ST-consistency of a PWFA X apparently requires that the sequence $(T_i(X))_{i=0}^{\infty}$ converges with respect to the function $h^*(m)$.

4.3 Alternative definitions of the computed set

Instead of using the definition of $S(X)$ as given in equation 4.2 for some PWFA X over some semiring $\mathcal{S}$, the following definitions for $S(X)$ could also be chosen. These definitions are all incompatible with the one we have chosen in the sense that they change the generative power of the automaton model.

1. $S(X) = \overline{\liminf}_{i\to\infty} S_i(X)$. This definition would reduce the power of PWFA with the alphabet $\Sigma = \{1\}$ to the generation of sets that are either empty or contain a single point only. The construction of a PWFA computing the unit circle given in [4] shows that this is not the case for the definition we are using.

2. $S(X) = \bigcup_{w\in\Sigma^\omega} S(w)$ where

$$S(w) = \begin{cases} \{\lim_{i\to\infty} f_X(a_1 \dots a_i)\} & \text{if } \lim_{i\to\infty} f_X(a_1 \dots a_i) \text{ exists} \\ \emptyset & \text{otherwise} \end{cases} \tag{4.10}$$

   for each $w = a_1 a_2 \ldots \in \Sigma_X^*$. As above, this definition would also reduce the computational power of PWFA over the unary alphabet to the computation of either the empty set or sets containing a single point only.

3. $S(X) = T(X)$. If we choose this definition, then it is impossible to define a PWFA computing the empty set. If, as we conjecture, there is no effective algorithm that given an affine iterated function system (IFS, [8]) computes a point lying on the attractor of the IFS, then this definition would in addition make it impossible to provide an effective algorithm that transforms an IFS into a PWFA that computes the attractor of the IFS.

As all of the above alternative computed set definitions are incompatible with the one we have chosen, they would imply a different PWFA theory that we will not consider in this thesis. The definition under point 3 is however relevant in applications, as the known practical decoding algorithms all approximate $T(X)$ and not $S(X)$.

4.4 Examples

Figure 4.1 and Figure 4.2 show two simple examples of PWFA. We depict PWFA similar to NFA. Let X denote the PWFA shown in Figure 4.1. Instead of only labeling the edges with alphabet symbols, we also add the corresponding weights (e.g. the edge going from state 1 to state 2 with weight -0.6 for the symbol 1 is labeled by $1 : -0.6$). The values of the characteristic functions χ_I and χ_F, which we use when we depict an NFA, are substituted by the initial weight vectors and the final weights of the respective states (e.g. state 1 of X is labeled by 1 (the state number), $I : (1\ 0)$ (as $(1\ 0)^T$ is the first column of the matrix I_X) and $F : (1)$ (as $F_X(1) = 1$)). Both automata, the one shown in Figure 4.1 and the one depicted in Figure 4.2, have already been presented in [4]. Furthermore, both automata are ST-consistent. Note that in the case of the automaton shown in Figure 4.1, let it be denoted by X, we have an example of a PWFA such that

$$\{(x,y) | x,y \in \mathbb{R}, x^2 + y^2 = 1\} = \overline{\limsup_{j\to\infty}} S_{X_j} \neq \overline{\liminf_{j\to\infty}} S_{X_j} = \emptyset. \tag{4.11}$$

4.5 Discussion of the word function

If we just consider the word function f_X computed by a PWFA X over some semiring $\mathcal{S}$, then we are still considering a WFA over some other semiring, as the following lemma shows.

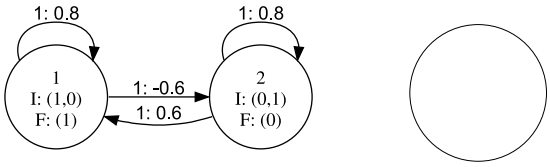


Figure 4.1: Unary alphabet PWFA computing the unit circle (left) and graphical representation of the computed set (right).

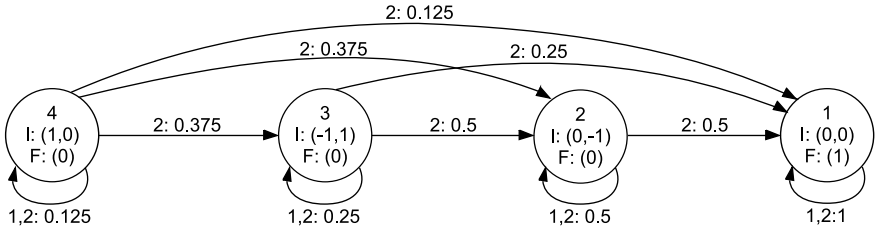


Figure 4.2: PWFA computing the set of points $\{(t^3-t^2, t^2-t) | t \in [0,1]\}$, which is depicted in Figure 4.3.

Figure 4.3: Graphical representation of the set computed by the PWFA shown in Figure 4.2.

Lemma 4.2 *Let $d \in \mathbb{N}^+$. Furthermore, let $\mathcal{S}$ be a semiring and let the addition and multiplication be defined component-wise in the semiring $\mathcal{S}^d$. Let X denote a PWFA of dimension d over $\mathcal{S}$ computing the function $f_X : \Sigma_X^* \mapsto \mathcal{S}^d$. There is a WFA Y over $\mathcal{S}^d$ using the same alphabet as X that computes $f_Y = f_X$.*

Proof: Let $a_d : \mathcal{S} \mapsto \mathcal{S}^d$ denote the function

$$a_d(c) = (\underbrace{\begin{array}{cccc} c & c & \dots & c \end{array}}_{d})^T \tag{4.12}$$

and $a_d^{-1} : \mathcal{S}^d \mapsto \mathcal{S}$

$$a_d^{-1}\left(\begin{array}{cccc} c_1 & c_2 & \dots & c_d \end{array}^T\right) = \begin{cases} c_1 & \text{if } c_1 = c_2 = \ldots = c_d \\ \text{undefined} & \text{otherwise} \end{cases} . \tag{4.13}$$

Y has the same set of states and alphabet as X, i.e. $Q_Y = Q_X$ and $\Sigma_Y = \Sigma_X$. Furthermore, we choose the elements of the matrix $A_{Y_k}(i,j)$ as $a_d(A_{X_k}(i,j))$ for each $k \in \Sigma, i, j \in Q$ and $F_Y(i) = a_d(F_X(i))$ for each $i \in Q$. Then we apparently have $a_d^{-1}((\mu_Y(w)F_Y)(i)) = (\mu_X(w)F_X)(i)$ for each $w \in \Sigma^*, i \in Q$. Choosing

$$I_Y = \left((I_X(1,1)I_X(2,1)\dots I_X(d,1))^T (I_X(1,2)I_X(2,2)\dots I_X(d,2))^T \dots\right) \tag{4.14}$$

completes the construction of Y. □

The converse statement also holds, as we show in the next lemma.

Lemma 4.3 *Let $d \in \mathbb{N}^+$. Furthermore, let $\mathcal{S}$ be a semiring and let the addition and multiplication be defined component-wise in the semiring $\mathcal{S}^d$. Let Y denote a WFA over $\mathcal{S}^d$ computing the function $f_Y : \Sigma_Y^* \mapsto \mathcal{S}^d$. There is a PWFA X of dimension d over $\mathcal{S}$ using the same alphabet as Y that computes the function $f_X = f_Y$.*

Proof: Assume that $Q_Y = \{1, \dots, n_Y\}$ for some $n_Y \in \mathbb{N}^+$ and $\Sigma_Y = \{1, \dots, l\}$ for some $l \in \mathbb{N}^+$. Let $p_i : \mathcal{S}^d \mapsto \mathcal{S}$ denote the projection onto component i for $i = 1, \dots, d$ and let $B_{i,j} \in \mathcal{S}^{d n_Y \times n_Y}$ denote the matrix

$$B_{i,j} = \begin{pmatrix} p_j(A_{Y_i}(1,1)) & p_j(A_{Y_i}(1,2)) & \dots & p_j(A_{Y_i}(1,n_Y)) \\ p_j(A_{Y_i}(2,1)) & p_j(A_{Y_i}(2,2)) & \dots & p_j(A_{Y_i}(2,n_Y)) \\ & \vdots & & \\ p_j(A_{Y_i}(n_Y,1)) & p_j(A_{Y_i}(n_Y,2)) & \dots & p_j(A_{Y_i}(n_Y,n_Y)) \end{pmatrix} \tag{4.15}$$

for $i = 1, \dots, l$ and $j = 1, \dots, d$. Further let

$$J_i = \begin{pmatrix} p_i(I_Y(1)) & p_i(I_Y(2)) & \dots & p_i(I_Y(n_Y)) \end{pmatrix} \tag{4.16}$$

for $i = 1, \dots, d$ and

$$G_i = \begin{pmatrix} p_i(F_Y(1)) & p_i(F_Y(2)) & \dots & p_i(F_Y(n_Y)) \end{pmatrix} \tag{4.17}$$

for $i = 1, \dots, d$. Then X can be constructed as

- $Q_X = \{1, \dots, d n_Y\}$,
- $\Sigma_X = \Sigma_Y$,

- $A_{X_i} = \begin{pmatrix} B_{i,1} & O & \dots & O \\ O & B_{i,2} & \dots & O \\ & & \vdots & \\ O & O & \dots & B_{i,d} \end{pmatrix}$ for $i = 1, \dots, l$ where O denotes the zero matrix in $\mathcal{S}^{n_Y \times n_Y}$,
- $I_X = \begin{pmatrix} J_1 & Q & \dots & Q \\ Q & J_2 & \dots & Q \\ & & \vdots & \\ Q & Q & \dots & J_d \end{pmatrix}$ where Q denotes the zero vector of $\mathcal{S}^{n_Y}$ and
- $F_X = \begin{pmatrix} G_1 & G_2 & \dots & G_d \end{pmatrix}^T$.

Then obviously X computes $f_X(w) = f_Y(w)$ for each $w \in \Sigma_X^*$. □

Combining the two lemmata we obtain the following theorem.

Theorem 4.1 *Let $d \in \mathbb{N}^+$. Furthermore, let $\mathcal{S}$ be a semiring and let the addition and multiplication be defined component-wise in the semiring $\mathcal{S}^d$. Let Σ be a finite alphabet. The function $f : \Sigma^* \mapsto \mathcal{S}^d$ can be computed by a PWFA of dimension d over $\mathcal{S}$, if and only if it can be computed by a WFA over $\mathcal{S}^d$.*

Remark 4.2 *Theorem 4.1 states that if we solely consider the word function, then the WFA and PWFA models have exactly the same expressiveness. The main difference between the two models is thus that we consider the set of vectors $S(X)$ for a PWFA X instead of the word function f_X.*

4.6 Initial and final normal form

We can consider the equivalence of PWFA under the following definition.

Definition 4.4 *Let X and Y be PWFA of dimension d over the semiring $\mathcal{S}$. We call X and Y*

1. (S-)equivalent*, if $S(X) = S(Y)$ and*
2. T-equivalent*, if $T(X) = T(Y)$.*

S-equivalence and T-equivalence are incomparable. Neither does S-equivalence imply T-equivalence nor does T-equivalence imply S-equivalence. The automata in Figure 4.4 and Figure 4.5 are S-equivalent but not T-equivalent, those in Figure 4.4 and Figure 4.6 are T-equivalent but not S-equivalent.

The next lemmata show that given a PWFA X, we can construct an S- and T-equivalent automaton X' that uses only a single initial state per dimension or a single final state. We also show that in general we cannot have both properties at once. We say that a PWFA is in initial normal form, if it has at most one initial state per dimension and in final normal form, if it has at most one final state. Unlike e.g the Jordan normal form for endomorphisms of real vector spaces, these normal forms do not uniquely describe the computed S or T sets, e.g. there are equivalent automata in initial or final normal form that are not identical up to a bijective remapping of states and alphabet symbols, see for example the automata shown in Figure 4.4 and Figure 4.5.

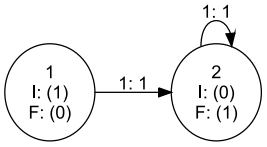


Figure 4.4: PWFA X of dimension 1 over $\mathbb{N}$ computing $S(X) = \{1\}$ and $T(X) = \{0,1\}$.

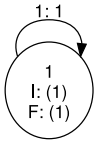


Figure 4.5: PWFA X of dimension 1 over $\mathbb{N}$ computing $S(X) = \{1\}$ and $T(X) = \{1\}$.

Lemma 4.4 *Let $X = (\{1\}, \Sigma, W, I = (i), F = (f))$ be a PWFA of dimension 1 over $\mathbb{N}$. Then the set $S(X)$ computed by X is either a subset of $\{0, if\}$ or infinite.*

The automaton X given in Figure 4.7 shows that two state automata over $\mathbb{N}$ can produce sets that cannot be produced by single state automata. It computes the set $S(X) = T(X) = \{a,b\}$ for arbitrary a and b, where we can assume $a \neq b$, $a \neq 0$ and $b \neq 0$. The next lemmata show that this is in general not possible, if we require the automaton to have only a single initial state per dimension and a single final state.

Lemma 4.5 *Let $X = (\{\{1,2\}, \{1,2,\ldots,l\}, W, I = (i_1 \;\; 0), F = (f_1 \;\; f_2)^T\}$ be a PWFA of dimension 1 over $\mathbb{N}$, where $0 \notin T(X) = \{t_1,t_2\} = S(X)$ for $t_1 \neq t_2$. Then $f_1 \neq 0$ and $f_2 \neq 0$.*

Remark 4.3 *The choice of state 1 as the single initial state is without loss of generality, we could choose state 2 as well.*

Remark 4.4 *Lemma 4.5 can easily be extended to automata using more states while requiring that the automaton has more than a single final state.*

Proof: Apparently $f_1 = 0$ or $i_1 = 0$ would imply $f_X(\varepsilon) = 0 \in T(X)$, so $f_1 \neq 0$ and $i_1 \neq 0$. If $A_k(1,2) = 0$ for each k, then $S(X) \neq \{t_1,t_2\}$ according to Lemma 4.4, because state 2 is never entered and we effectively have a single state automaton,

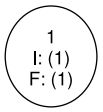


Figure 4.6: PWFA X of dimension 1 over $\mathbb{N}$ computing $S(X) = \{0\}$ and $T(X) = \{0,1\}$.

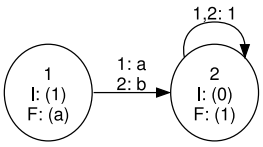


Figure 4.7: PWFA X of dimension 1 over $\mathbb{N}$ computing the set $S(X) = T(X) = \{a,b\}$ for arbitrary $a,b \in \mathbb{N}$.

so we assume that $A_k(1,2) \neq 0$ for at least one k. If $A_k(1,1) = 0$ for some k, then $f_2 \neq 0$ or $f_X(k) = 0 \in T(X)$. Thus we have at least one of $f_2 \neq 0$ or $A_k(1,1) \neq 0$ for all k. Assume $f_2 = 0$ and thus $A_k(1,1) \neq 0$ for all k. Then also $A_k(2,1) \neq 0$ for at least one k or again $S(X) \neq \{t_1,t_2\}$ according to Lemma 4.4, because state 2 could only be entered but not left and we again would effectively have a single state automaton. If $A_k(1,1) > 1$ for at least one k, then $T(X)$ is infinite, because the values appearing in state 1 are unbounded but bounded for each word length, i.e. $f_X(w)(1) \leq b(n)$ for each $n \in \mathbb{N}$, each $w \in \Sigma^n$ and a total function $b : \mathbb{N} \mapsto \mathbb{N}$, while arbitrarily large values can appear in state 1. Thus $A_k(1,1) \leq 1$ for all k and consequently $A_k(1,1) = 1$ for all k. If $A_k(1,2)A_m(2,1) > 1$ for some pair of symbols $(k,m) \in \Sigma \times \Sigma$, then $T(X)$ is infinite, thus $A_k(1,2) \leq 1$ and $A_k(2,1) \leq 1$ for each k. $T(X)$ is for similar reasons also infinite if $A_k(2,2) > 1$ for some k. Summarizing we now have that $A_k(r,s) \leq 1$ for each $k \in \Sigma, r,s \in Q$ and $A_k(1,1) = 1$ for all k. This means that the values in state 1 are monotonously increasing, as the automaton can only add positive numbers to what is already contained in state 1. There is a pair of symbols $(k,m) \in \Sigma \times \Sigma$ for which $A_k(1,2)A_m(2,1) \neq 0$, which implies that reading the string km increases the value contained in state 1 by some positive integer, so $T(X)$ again is an infinite set. This means that the assumption of $f_2 = 0$ made above was wrong. □

Lemma 4.6 *Let $X = \{\{1,2\},\{1,2,\ldots,l\},W,I = (i_1 \;\; i_2),F = (f_1 \;\; 0)^T\}$ be a PWFA of dimension 1 over $\mathbb{N}$, where $0 \notin T(X) = \{t_1,t_2\} = S(X)$ for $t_1 \neq t_2$. Then $i_1 \neq 0$ and $i_2 \neq 0$.*

Remark 4.5 *The choice of state 1 as the single final state is without loss of generality, we could choose state 2 as well.*

Remark 4.6 *Lemma 4.6 can easily be extended to automata using more states while requiring that the automaton has more than a single initial state.*

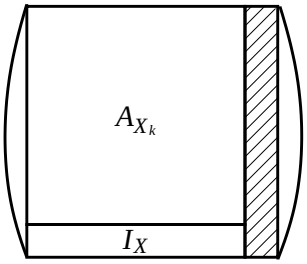
Proof: Apply the argumentation in the proof of Lemma 4.5 after transposing the transition matrices of X and swapping I and F^T. □

Lemma 4.7 *Let X be a PWFA of dimension d over the semiring $\mathcal{S}$. There is an S- and T-equivalent PWFA Z that uses exactly one initial state per dimension.*

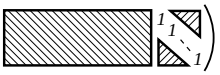
Proof: Z can be constructed as

- $Q_Z = (1,2,\ldots,|Q_X|,|Q_X|+1,\ldots,|Q_X|+d\}$,
- $\Sigma_Z = \Sigma_X$,

- The A_{Z_i} are block matrices of the following form, where striped areas are blocks of zeroes:

$A_{Z_k} =$  for $k = 1,2,\ldots,l$,

- I_Z is a matrix of the following structure (striped areas are zero areas as above):

$I_Z =$ 

  All rows are unit vectors, where the initial distribution for dimension i has the value 1 at the state $|Q(X)| + |Q(Y)| + i$ and is zero otherwise.

- ${F_Z}^T$ is the vector $({F_X}^T \underbrace{(I_X F_X)^T}_{d})$

In comparison to X, the word function of Z is delayed by one symbol, i.e. for each $a \in \Sigma$ and each $w \in \Sigma^*$ the equation

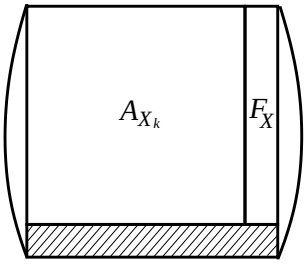
$$f_Z(aw) = f_X(w) \tag{4.18}$$

holds, which implies that $S(Z) = S(X)$. As $f_Z(\varepsilon) = f_X(\varepsilon)$, Z is also T-equivalent to X. □

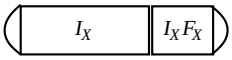
Lemma 4.8 *Let X be a PWFA of dimension d over the semiring $\mathcal{S}$. There is an S- and T-equivalent PWFA Z that uses exactly one final state.*

Proof: Z can be constructed as

- $Q_Z = (1,2,\ldots,|Q_X|,|Q_X|+1\}$,
- $\Sigma_Z = \Sigma_X$,
- The A_{Z_i} are block matrices of the following form, where striped areas are blocks of zeroes:

$A_{Z_k} =$  for $k = 1,2,\ldots,l$,

- I_Z is a matrix of the following structure:



The first $|Q_X|$ columns contain the matrix I_X, the $|Q_X|+1$-th column is $I_X F_X$.

- ${F_Z}^T$ is the vector $(\underbrace{0\ 0\ldots 0}_{|Q_X|}\ 1)$.

In comparison to X, the word function of Z is shifted by one symbol, i.e. for each $a \in \Sigma$ and each $w \in \Sigma^*$ the equation

$$f_Z(wa) = f_X(w) \tag{4.19}$$

holds, which implies that $S(Z) = S(X)$. As $f_Z(\varepsilon) = f_X(\varepsilon)$, Z is also T-equivalent to X. □

The critical part in the construction of the initial and final normal form automata is the behavior of the word function on the empty word. If we neglect this, we are able to construct an (S-)equivalent automaton X' for each automaton X that uses only a single initial state per dimension and a single final state. This however will in general not yield a T-equivalent automaton.

4.7 PWFA alphabet cardinality

In this section we show that every PWFA computable set can be computed by a PWFA with an alphabet cardinality of 2. This result was first published in [90].

Theorem 4.2 *Let X be a PWFA of dimension d over the semiring $\mathcal{S}$. There is a PWFA Y of dimension d over $\mathcal{S}$ with alphabet cardinality 2 such that $S(Y) = S(X)$ and $T(Y) = T(X)$ hold.*

Proof: Assume that $Q_X = \{1,\ldots,n\}$ for some $n \in \mathbb{N}^+$. Furthermore, assume without loss of generality that $\Sigma_X = \{0,\ldots,2^k-1\}$ for some positive natural number k. If this does not hold, we can add alphabet symbols to X and assign any of the transition matrices already present in the automaton to them without changing $S(X)$ and $T(X)$. We start the alphabets at 0 instead of 1 in this proof because we want to interpret input words using the functions $q[2]$ and $q[2^k]$. The alphabet Σ_Y, as postulated, is the set $\{0,1\}$. The automaton Y has $(2^k-1)n$ states, i.e. 2^k-1 times as many as X. We imagine these states as grouped in groups of size n, where the groups are arranged to form a complete binary tree of depth $k-1$. Let $Q_k(w) : \Sigma_Y^* \mapsto \Sigma_Y^*$ denote the function that maps each word $w \in \Sigma_Y^*$ to the longest prefix p of w such that $|p|$ is a multiple of k. For each natural number j such that $j \geq 2$ let further $q[j]^*(x) : \mathbb{Q}_U \mapsto \wp(\{0,\ldots,j-1\}^*)$ be given by

$$q[j]^*(x) = \{w | w \in \{0,\ldots,j-1\}^* \text{ and } q[j](w) = x\} \tag{4.20}$$

for each $x \in \mathbb{Q}_U$, i.e. the set of words that $q[j]$ maps to x, let $\mathcal{X}[j](x,i) : \mathbb{Q}_U \times \mathbb{N} \mapsto \{0,\ldots,j-1\}^i$ be defined by

$$\mathcal{X}[j](x,i) = q[j]^*(x) \cap \{0,\ldots,j-1\}^i \tag{4.21}$$

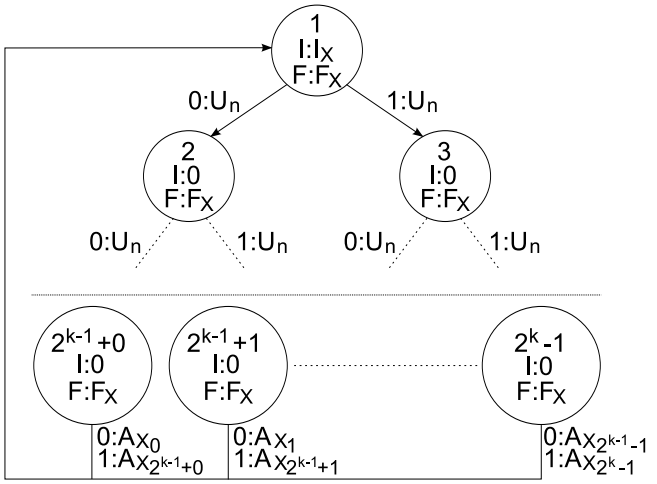


Figure 4.8: Sketch of the automaton Y constructed in the proof of Theorem 4.2. Each vertex represents a group of n states.

for each $x \in \mathbb{Q}_U, i \in \mathbb{N}$ and let $q[j]^{-1}(x,i) : \mathbb{Q}_U \times \mathbb{N} \mapsto \{0,\ldots,j-1\}^i$ be defined by

$$q[j]^{-1}(x,i) = \begin{cases} \text{the single element of } \mathcal{X}[j](x,i) & \text{if } \mathcal{X}[j](x,i) \neq \emptyset \\ \text{undefined} & \text{otherwise} \end{cases} \tag{4.22}$$

for each $x \in \mathbb{Q}_U$ and each natural number i. Then we construct the transition matrices A_{Y_1} and A_{Y_2} in such a way that

$$f_Y(w) = f_X\left(q[2^k]^{-1}\left(q[2](Q_k(w)), \frac{|Q_k(w)|}{k}\right)\right) \tag{4.23}$$

for each $w \in \{0,1\}^*$, i.e. one transition of X is simulated by k steps of Y. If we define f_Y in this way, then apparently $T(Y) = T(X)$ and $S(Y) = S(X)$. Let $B_m(i,j,C) \in \mathcal{S}^{nm\times nm}$ for each positive natural number m, $1 \leq i,j \leq m$, and $C \in \mathcal{S}^{n\times n}$ denote the matrix consisting of m rows and m columns of $n \times n$ blocks such that the block at block position (i,j) is the matrix C and all other elements are zero. Further let U_n denote the identity matrix in $\mathcal{S}^{n\times n}$. Then we define the transition matrices A_{Y_0} and A_{Y_1} by writing

$$A_{Y_0} = \sum_{i=1}^{k-1}\sum_{j=0}^{2^{i-1}} B_{2^k-1}\left(2^i + j, i, U_n\right) + \sum_{i=0}^{\frac{k}{2}-1} B_{2^k-1}\left(2^{k-1}+i, 1, A_{X_i}\right) \tag{4.24}$$

and

$$A_{Y_1} = \sum_{i=1}^{k-1}\sum_{j=0}^{2^{i-1}} B_{2^k-1}\left(2^i + 2^{i-1} + j, i, U_n\right) + \sum_{i=0}^{\frac{k}{2}-1} B_{2^k-1}\left(2^{k-1}+i, 1, A_{X_{i+\frac{k}{2}}}\right) \quad . \tag{4.25}$$

Further we define $I_Y \in \mathcal{S}^{d\times(2^k-1)n}$ as the matrix that equals I_X in the first n columns and is zero otherwise and $F_Y = (F_X^T \ldots F_X^T)^T$. If we define Y in such a way, then equation 4.23 holds. The automaton Y is sketched in Figure 4.8. □

Chapter 5

Operations and decision problems on PWFA computable sets

5.1 Closure under set primitives

The closure of $\mathcal{D}(\mathbb{R},d)$ under set union, invertible affine transformation and regular restriction for each $d \in \mathbb{N}^+$ was first presented in [89] .

In this section we will show that $\mathcal{D}(\mathcal{S},d)$ is closed under set union and regular restriction for each semiring $\mathcal{S}$ and each dimension $d \in \mathbb{N}^+$. Furthermore, we will show that for each dimension $d \in \mathbb{N}^+$ and each topological semiring $\mathcal{S}$ the set $\mathcal{D}(\mathcal{S},d)$ is closed under translations, linear transformations and affine transformations being topological automorphisms on $\mathcal{S}^d$ (i.e. homeomorphisms from $\mathcal{S}^d$ onto itself) and that the set of sets computable by ST-consistent PWFA over the complex numbers is closed under arbitrary affine transformations. We will also show that the set of sets computable by PWFA X over the complex numbers, for which $S(X) = T(X)$ holds, is closed under iterated affine transformation.

We will implement the single and iterated application of linear transformations to the set computed by a PWFA by manipulating the initial distribution of the automaton. We therefore introduce a short notation to describe manipulations of the initial distribution of a PWFA.

Definition 5.1 *Let $X = (Q = \{1,\ldots,n\},\Sigma,W,I,F)$ be a PWFA of dimension d over the semiring $\mathcal{S}$ and let $J \in \mathcal{S}^{d\times n}$. Then $\Xi[J](X)$ denotes the automaton (Q,Σ,W,J,F), i.e. the automaton X, where the initial distribution I of X has been substituted by the matrix J.*

5.1.1 Set union

We show that the set $\mathcal{D}(\mathcal{S},d)$ is closed under the set union operation for each semiring $\mathcal{S}$ and each dimension $d \in \mathbb{N}^+$. An example of this operation is depicted in Figure 5.1.

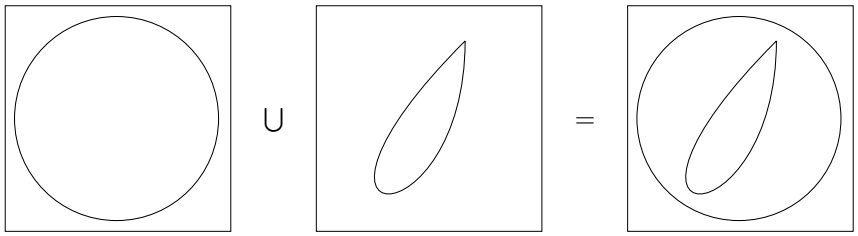


Figure 5.1: Closure under set union: if there are automata X and Y computing the sets $S(X)$ (a circle, left) and $S(Y)$ (a parametric curve, middle), then there is an automaton Z computing $S(Z) = S(X) \cup S(Y)$ (a circle and a parametric curve, right).

Theorem 5.1 *Let X and Y be PWFA of dimension d over the semiring $\mathcal{S}$. Then there exists a PWFA Z of dimension d over $\mathcal{S}$ such that $S(Z) = S(X) \cup S(Y)$ and $T(Z) = T(X) \cup T(Y)$. Moreover, if $\Sigma_X = \Sigma_Y = \{1\}$, then there exists a PWFA Z of dimension d over $\mathcal{S}$ such that $S(Z) = S(X) \cup S(Y)$, $T(Z) = T(X) \cup T(Y)$ and $\Sigma_Z = \{1\}$.*

Proof: Assume that $Q_X = \{1, \ldots, n\}$ for some $n \in \mathbb{N}^+$ and $Q_Y = \{1, \ldots, m\}$ for some $m \in \mathbb{N}^+$. We consider two cases.

Case 1: $|\Sigma_X| + |\Sigma_Y| > 2$. Assume without loss of generality that $\Sigma_X = \Sigma_Y = \{1, \ldots, l\}$ for some $l \in \mathbb{N}, l \geq 2$. Then Z can be defined as

- $Q_Z = \{1, \ldots, n+m+d\}$
- $\Sigma_Z = \Sigma_X$
- The A_{Z_i} are block matrices of the following form where striped areas are blocks of zeroes:

$$A_{Z_i} = \begin{cases} \begin{pmatrix} A_{X_i} & & \\ & A_{Y_i} & \\ I_X & & \end{pmatrix} & \text{if } i \text{ is odd} \\ \begin{pmatrix} A_{X_i} & & \\ & A_{Y_i} & \\ & I_Y & \end{pmatrix} & \text{if } i \text{ is even} \end{cases}$$

- I_Z is a matrix of the following structure (striped areas are zero areas as above):

$$I_Z = \left(\left(\quad \begin{matrix} & & 1 \\ & 1 & \\ 1 & & \end{matrix} \right) \right) \tag{5.1}$$

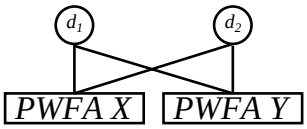


Figure 5.2: Structure of the union automaton for the PWFA X and Y in the case of a dimension of two.

All rows are unit vectors, where the initial distribution for dimension i has the value 1 at the state $n+m+i$ and is zero otherwise.

- $F_Z{}^T$ is the vector $(F_X{}^T F_Y{}^T \underbrace{(I_X F_X)^T}_{d})$.

The produced automaton Z for the example of $d = 2$ has the structure shown in Figure 5.2, where d_i in the figure corresponds to state number $n+m+i$ in the automaton Z. The first symbol of a word $w = a_1 a_2 \ldots a_p \in \Sigma_Z^*$ decides, which of the sets $S(X)$ or $S(Y)$ the automaton computes on words with $|w| \geq 1$. If a_1 is odd, then $f_Z(w) = f_X(a_2 a_3 \ldots a_p)$, if a_1 is even, then $f_Z(w) = f_Y(a_2 a_3 \ldots a_p)$. Thus

$$S_k(Z) = \begin{cases} \{I_X F_X\} \text{ for } k = 0 \\ S_{k-1}(X) \cup S_{k-1}(Y) \text{ for } k > 0 \end{cases} \tag{5.2}$$

This leads to

$$S_{\geq k}(Z) = \begin{cases} \bigcup_{i=0}^{\infty} (S_i(X) \cup S_i(Y)) & \text{for } k = 0 \\ \bigcup_{i=k-1}^{\infty} (S_i(X) \cup S_i(Y)) & \text{for } k > 0 \end{cases} \tag{5.3}$$

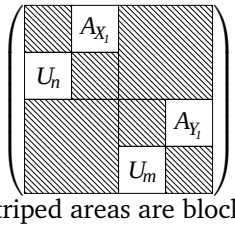
As the vector $I_X F_X$ is an element of $S_0(X)$, the case for $k = 0$ in (5.3) is correct and so $S_{\geq 0}(Z) = S_{\geq 1}(Z)$. The set computed by Z is

$$\begin{aligned} S(Z) &= \bigcap_{j=0}^{\infty} \overline{S_{\geq j}(Z)} = \bigcap_{j=1}^{\infty} \overline{S_{\geq j}(Z)} = \bigcap_{j=0}^{\infty} \overline{\bigcup_{k=j}^{\infty} (S_k(X) \cup S_k(Y))} \\ &= \bigcap_{j=0}^{\infty} \overline{\bigcup_{k=j}^{\infty} S_k(X) \cup \bigcup_{k=j}^{\infty} S_k(Y)} \\ &= \bigcap_{j=0}^{\infty} \overline{\bigcup_{k=j}^{\infty} S_k(X)} \cup \overline{\bigcup_{k=j}^{\infty} S_k(Y)} \\ &= \bigcap_{j=0}^{\infty} \left(\overline{\bigcup_{k=j}^{\infty} S_k(X)} \cup \overline{\bigcup_{k=j}^{\infty} S_k(Y)} \right) \\ &= \bigcap_{j=0}^{\infty} \overline{\bigcup_{k=j}^{\infty} S_k(X)} \cup \bigcap_{j=0}^{\infty} \overline{\bigcup_{k=j}^{\infty} S_k(Y)} = S(X) \cup S(Y) \ . \end{aligned} \tag{5.4}$$

Further $T(Z) = \overline{S_{\geq 0}(Z)} = \overline{S_{\geq 0}(X) \cup S_{\geq 0}(Y)} = \overline{S_{\geq 0}(X)} \cup \overline{S_{\geq 0}(Y)} = T(X) \cup T(Y)$.

Case 2: $|\Sigma_X| = |\Sigma_Y| = 1$: Z can be defined as

- $Q_Z = \{1, \ldots, 2(n+m)\}$
- $\Sigma_Z = \{1\}$
- $I_{Z_k} = (I_{X_k} \ \underbrace{0 \ 0 \ldots 0}_{n} \ \underbrace{0 \ 0 \ldots 0}_{m} \ I_{Y_k})$ for $k = 1, 2, \ldots d$
- $F_Z = (F_X{}^T \ \underbrace{0 \ 0 \ldots 0}_{n} \ F_Y{}^T \ \underbrace{0 \ 0 \ldots 0}_{m})^T$

- $A_{Z_1} =$ 

where U_k denotes the $k \times k$ identity matrix and striped areas are blocks of zero.

Z computes the function

$$f_Z(1^k) = \begin{cases} f_X(1^{k/2}) \text{ if } k \text{ is even} \\ f_Y(1^{(k-1)/2}) \text{ if } k \text{ is odd.} \end{cases} \tag{5.5}$$

Thus $S(Z) = S(X) \cup S(Y)$ and $T(Z) = T(X) \cup T(Y)$.

□

Definition 5.2 *Let X and Y be PWFA over the same semiring $\mathcal{S}$ such that $d_X = d_Y$. Then we denote the automaton Z constructed in the proof of Theorem 5.1 to compute $S(Z) = S(X) \cup S(Y)$ by $X \cup Y$.*

A few further remarks concerning the construction are:

- The restriction to merging only automata with an equal dimension in the theorem could be overcome by adding null dimensions to the automaton with the smaller dimension.

- If we merge two automata X and Y of dimension d, where $|\Sigma_X| = |\Sigma_Y| \geq 2$, then the construction always yields an automaton Z that has $|Q_X| + |Q_Y| + d$ states. If we however for instance merge two automata X and Y that only differ in their initial distribution, then there obviously is an automaton Z computing $S(Z) = S(X) \cup S(Y)$ such that Z has only $|Q_X| + d$ states. Thus the automata constructed in the proof are in some cases unnecessarily large, especially if we merge two automata that have a similar structure.

- Let $\mathcal{X} = \{X_1, \ldots, X_k\}$ be a finite set of PWFA over the semiring $\mathcal{S}$ such that $d_{X_1} = \ldots = d_{X_k}$ and $|\Sigma_{X_1}| = \ldots = |\Sigma_{X_k}|$. If we want to obtain a PWFA Z computing $S(Z) = S(X_1) \cup \ldots \cup S(X_k)$, then this should be done using the function PWFA_MERGE shown in Algorithm 1 (let the obtained automaton be denoted by Z_1) and not by computing $Z_2 = (\ldots(X_1 \cup X_2) \cup X_3 \ldots) \cup X_k$. The initial states of the sub-automata $X_1, \ldots, X_k$ inside Z_1 can all be reached from the initial states of Z_1 by taking roughly $\lceil \log_2(k) \rceil$ edges. In Z_2 the initial states of the contained sub-automaton X_1 are not reached until until the automaton has made $k-2$ transitions. In practice k can get large, we frequently reach values of k above 10000. Some PWFA decoding algorithms are based on evaluating the function f_X computed by a given PWFA X for all words of a certain length N. While word lengths of about $\lceil \log_2(10000) \rceil = 13$, which for instance means we consider 2^{13} words if we use an alphabet size of 2, can be handled in practice, this is not the case for word lengths in the order of 10000, where we cannot handle k^{10000} words for any $k > 1$.

Input : Non-empty set of $\mathcal{X} = \{X_1, \ldots, X_k\}$ of k PWFA over some semiring $\mathcal{S}$ such that $\Sigma_{X_1} = \ldots = \Sigma_{X_k}$ and $d_{X_1} = \ldots = d_{X_k}$
Output: Automaton Z computing $S(Z) = S(X_1) \cup \ldots \cup S(X_k)$

```
while |X| > 1 do
    j ← |X|
    // convert set X to sequence
    (X_1, ..., X_j) ← X
    Y ← {}
    for i = 1, ..., ⌊j/2⌋ do
        Y ← Y ∪ {X_{2i-1} ∪ X_{2i}}
    if j is odd then
        Y ← Y ∪ {X_j}
    X ← Y
// convert set X to sequence
(X_1) ← X
Z ← X_1
return Z
```

Algorithm 1: PWFA_MERGE($\mathcal{X}$): Merge a set $\mathcal{X}$ of PWFA.

The set computed by a PWFA can be decomposed into a finite union of PWFA computable sets, as the following theorem shows. This can be understood as an inverse to the union construction given above.

Theorem 5.2 *Let $X = (Q, \Sigma, W, I, F)$ denote a PWFA of dimension d over the semiring $\mathcal{S}$ and let $w_1, \ldots, w_k \in \Sigma^*, |w_1| \leq |w_2| \leq \ldots \leq |w_k| = m$ for some positive natural number k be a complete prefix code (implying that no word w_i is a prefix of any word w_j for $i \neq j$ and that every word $w \in \Sigma^m\Sigma^*$ has exactly one of the w_i as a prefix). Then*

$$S(X) = \bigcup_{i=1}^{k} S(\Xi[I\mu(w_i)](X)) \quad . \tag{5.6}$$

Proof: Let $\mu = \mu_X$.
Assume that $v \in S(X)$. Then there is a sequence of words $x_1, \ldots$ of increasing length in $\Sigma^m\Sigma^*$ such that $\lim_{i\to\infty} f_X(x_i) = v$. As the words $w_1, \ldots, w_k$ form a complete prefix code, each x_i is prefixed by exactly one of the w_j. There are infinitely many x_i. Thus at least one of the w_j is a prefix to infinitely many x_i. Let one such w_j be denoted by w_α and let $(x_i') = x_1', \ldots$ denote the subsequence of (x_i) consisting of the words x_i which have the prefix w_α. Let (x_i'') denote the sequence obtained from (x_i') by removing the prefix w_α from the beginning of each word. Then $\lim_{i\to\infty} f_{\Xi[I\mu(w_\alpha)](X)}(x_i'') = v$ and thus $v \in S(\Xi[I\mu(w_\alpha)](X))$.
Now let $v \in S(\Xi[I\mu(w_\alpha)](X))$ for some $\alpha \in \{1, \ldots, k\}$. Then there is a sequence of words of increasing length $x_1, \ldots$ in Σ^* such that $\lim_{i\to\infty} f_{\Xi[I\mu(w_\alpha)](X)}(x_i) = v$ and thus $\lim_{i\to\infty} f_X(w_\alpha x_i) = v$. Consequently $v \in S(X)$. □

The result given in Theorem 5.2 is valuable in applications. In many cases we can decompose the set $S(X)$ computed by a PWFA X into smaller sets $S(X_1)$, $\ldots, S(X_k)$ computed by the PWFA $X_1, \ldots, X_k$ respectively, where the evaluation of each X_i requires a much smaller amount of physical resources (e.g. memory) then the evaluation of X.

5.1.2 Translation

Theorem 5.3 *Let X be a PWFA of dimension d over the semiring $\mathcal{S}$ and let $g(x) = x + c$ for $c \in \mathcal{S}^d$ be a topological automorphism on $\mathcal{S}^d$. There is a PWFA Z of dimension d over $\mathcal{S}$ computing the set*

$$S(Z) = S(X) + c = \{v | v = x + c \text{ for some } x \in S(X)\} \quad . \tag{5.7}$$

Proof: Assume that $Q_X = \{1, \ldots, n\}$ for some $n \in \mathbb{N}^+$. Z is obtained by adding one state $n+1$ to X, setting $A_{Z_a}(n+1, n+1) = 1$ for each $a \in \Sigma$, $A_{Z_a}(n+1, i) = A_{Z_a}(i, n+1) = 0$ for each $a \in \Sigma, i = 1, \ldots, n$, column $n+1$ in I_Z to c and the final value of state $n+1$ to 1. Let $v_1, v_2, \ldots$ be a sequence of words of increasing length in Σ^*. As $g(x) = x + c$ is a homeomorphism, the sequence $f_X(v_1), f_X(v_2), \ldots$ converges, if and only if $f_Z(v_1) = f_X(v_1) + c, f_Z(v_2) = f_X(v_2) + c, \ldots$ converges. Thus $S(Z) = S(X) + c$. □

An analogous proof can be used for the following theorem.

Theorem 5.4 *Let X be a PWFA of dimension d over the semiring $\mathcal{S}$ endowed with the discrete topology and let $g(x) = x + c$ for $c \in \mathcal{S}^d$ be a bijective function on $\mathcal{S}^d$. There is a PWFA Z of dimension d over $\mathcal{S}$ computing the set*

$$S(Z) = S(X) + c = \{v | v = x + c \text{ for some } x \in S(X)\} \quad . \tag{5.8}$$

As every translation on a complex vector space using the standard topology is a topological automorphism on the space, the set $\mathcal{D}(\mathbb{C}, d)$ is closed under arbitrary translation for each $d \in \mathbb{N}^+$.

5.1.3 Linear transformation

We will show that the linear transformation of a PWFA computed set is PWFA computable, if at least one of two constraints holds. The first constraint is that the linear transformation is a topological automorphism on the considered topological space, which means that we put a constraint on the linear transformation, but not on the considered automaton. The second constraint is that the considered automaton over some complete metric space and semiring $\mathcal{K} \subset \mathbb{C}$ is *ST*-consistent. Here we put a constraint on the automaton and the structures it works on, but not on the linear transformation that is applied.

Theorem 5.5 *Let X be a PWFA of dimension d over the semiring $\mathcal{S}$ and let $g(x) = Bx$ be linear transformation on $\mathcal{S}^d$, where g is a topological automorphism on $\mathcal{S}^d$. Then the PWFA $Z = \Xi[BI_X](X)$ computes $S(Z) = g(S(X))$.*

Proof: Obviously Z computes

$$S(Z) = \bigcap_{j=0}^{\infty} \overline{\bigcup_{k=j}^{\infty} \{B f_X(w) | w \in \Sigma_X^k\}} \tag{5.9}$$

and we have to show that $S(Z) = g(S(X))$. Let $v_1, v_2, \ldots$ be a sequence of words of increasing length in Σ_X^*. g is a homeomorphism. Thus $\lim_{i \to \infty} f_X(v_i)$ exists, if and only if $\lim_{i \to \infty} f_Z(v_i)$ exists. If both exist, then $\lim_{i \to \infty} f_Z(v_i) = g(\lim_{i \to \infty} f_X(v_i))$. □

An analogous proof can be used for the following theorem for semirings endowed with the discrete topology.

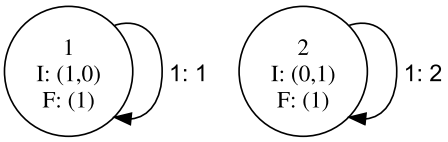


Figure 5.3: Real unary alphabet PWFA of dimension 2 computing the empty set.

Theorem 5.6 *Let X be a PWFA of dimension d over the semiring $\mathcal{S}$ endowed with the discrete topology and let $g(x) = Bx$ be a bijective linear transformation on $\mathcal{S}^d$. Then the PWFA $Z = \Xi[BI_X](X)$ computes $S(Z) = g(S(X))$.*

Theorem 5.7 *Let $(\mathcal{K}^d, m)$ for some $d \in \mathbb{N}^+$ denote a complete metric space, where $\mathcal{K} \subset \mathbb{C}$ is a semiring and the metric m is given as $m(x,y) = |x-y|$ for $x,y \in \mathbb{C}$ (i.e. the metric induced by the L^2 or Euclidean norm). Furthermore, let X be an ST-consistent PWFA of dimension d over $\mathcal{K}$ and $g(x) = Bx$ a linear transformation on $\mathcal{K}^d$. Then the PWFA $Z = \Xi[BI_X](X)$ computes $S(Z) = g(S(X))$.*

Proof: We have $f_Z(w) = Bf_X(w)$ by construction for all $w \in \Sigma^*$. Let $v_1, v_2, \ldots$ be a sequence of words of increasing length in Σ_X^*. If $\lim_{i\to\infty} f_X(v_i)$ exists, then clearly this is also the case for

$$\lim_{i\to\infty} f_Z(v_i) = \lim_{i\to\infty} g(f_X(v_i)) \tag{5.10}$$

because g is continuous. Thus $k \in S(X)$ implies $g(k) \in S(Z)$. As g is an arbitrary (i.e. not necessarily invertible) linear transformation, it may convert non-convergent sequences of vectors in $\mathcal{K}^d$ to convergent sequences. Thus assume that $\lim_{i\to\infty} f_X(v_i)$ does not exist, but $\lim_{i\to\infty} f_Z(v_i)$ does exist. The automaton X is *ST*-consistent. This implies that for increasing word length i all members of S_i become arbitrarily close to vectors contained in $S(X)$, i.e.

$$\lim_{i\to\infty} \max_{a\in S_i(X)} \min_{b\in S(X)} \{m(a,b)\} = 0 \ . \tag{5.11}$$

This in turn means that the limit $\lim_{i\to\infty} f_Z(v_i)$ is also produced as the limit of a sequence $(f_Z(v'_i))_{i=1}^{\infty}$ for a sequence of words $(v'_i)_{i=1}^{\infty}$ of increasing length such that $\lim_{i\to\infty} f_X(v'_i)$ exists and $\lim_{i\to\infty} f_Z(v'_i) = \lim_{i\to\infty} g(f_X(v'_i))$. □

The automaton with one label shown in Figure 5.3, let it be denoted by X, computes the empty set. X produces the vector $(1\ 2^k)^T$ for the input word with length $k \in \mathbb{N}$, thus there is no converging sequence of words of increasing length for this automaton. The theorems above make no statement for the application of the affine transformation

$$g(x) = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} x \tag{5.12}$$

to X, because g is not a homeomorphism (it does not allow a (continuous) inverse) and X is not *ST*-consistent. However, $g(S(X)) = \emptyset$ is clearly an element of $\mathcal{D}(\mathbb{C},2)$. It is an open problem, whether $\mathcal{D}(\mathbb{C},d)$ is closed under arbitrary linear transformation for each d.

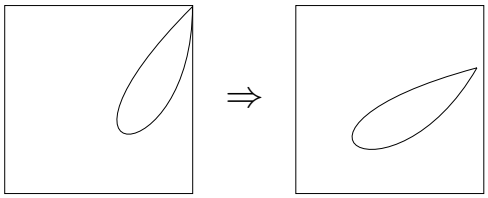


Figure 5.4: Closure under affine transformation: The curve on the left, computed by the automaton shown in Figure 4.2, is subject to a rotation followed by a translation to obtain the curve on the right.

5.1.4 Affine transformation

An affine transformation is a combination of a linear transformation and a translation. Thus we can state the following theorems based on the results given above.

Theorem 5.8 *Let X be a PWFA of dimension d over the semiring $\mathcal{S}$ and let $g(x) = Ax + b$ be an affine transformation on $\mathcal{S}^d$ such that g is a topological automorphism. Then there exists a PWFA Z of dimension d over $\mathcal{S}$ computing the set $S(Z) = g(S(X))$.*

Theorem 5.9 *Let X be a PWFA of dimension d over the semiring $\mathcal{S}$ endowed with the discrete topology and let $g(x) = Ax + b$ be an invertible affine transformation on $\mathcal{S}^d$. Then there exists a PWFA Z of dimension d over $\mathcal{S}$ computing the set $S(Z) = g(S(X))$.*

Theorem 5.10 *Let $(\mathcal{K}^d, m)$ for some $d \in \mathbb{N}^+$ denote a complete metric space, where $\mathcal{K} \subset \mathbb{C}$ is a semiring and the metric m is given as $m(x,y) = |x-y|$ for $x, y \in \mathbb{C}$ (i.e. the metric induced by the L^2 or Euclidean norm). Furthermore, let X be a PWFA of dimension d over $\mathcal{K}$ such that X is ST-consistent. Let $g(x) = Ax + b$ be an affine transformation of $\mathcal{K}^d$. Then there exists a PWFA Z of dimension d over $\mathcal{K}$ such that $S(Z) = g(S(X))$.*

An example of the closure under affine transformation is shown in Figure 5.4.

5.1.5 Iterated affine transformation

Definition 5.3 *Let $d \in \mathbb{N}^+$. Furthermore, let $\mathcal{S}$ be a semiring and $g_1, g_2, \ldots, g_r$ a finite sequence of affine transformations on $\mathcal{S}^d$. Let $\Gamma = \{1, \ldots, r\}$. Then we define*

$$\nu(u)(x) = \begin{cases} x & \text{if } u = \varepsilon \\ (g_{a_1} \circ g_{a_2} \circ \ldots g_{a_n})(x) & \text{if } u \neq \varepsilon \end{cases} \tag{5.13}$$

for each $x \in \mathcal{S}^d, u = a_1 a_2 \ldots a_n \in \Gamma^*$ and

$$\nu(u)(S) = \bigcup_{y \in S} \{\nu(u)(y)\} \tag{5.14}$$

for each $S \subset \mathcal{S}^d, u \in \Gamma^*$. Further we call

$$\nu(S) = \bigcup_{u \in \Gamma^*} \nu(u)(S) \tag{5.15}$$

the iterated image of S under $g_1, \ldots, g_r$ for each $S \subset \mathcal{S}^d$.

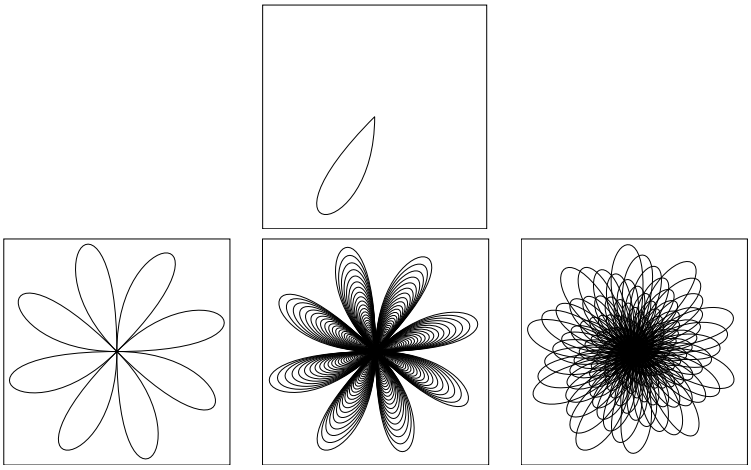


Figure 5.5: Closure under iterated affine transformation: The curve on the top is computed by the automaton shown in Figure 4.2. The three images in the bottom row show iterated images of the curve, where each iterated image is based on a different single map. The map used on the left is a rotation by 45°. The middle image is produced by a 45° rotation composed with a scaling transformation for a scaling factor smaller than 1. The picture shown on the right is based on a rotation by an angle of $\text{acos}(\frac{4}{5})$ composed by the same scaling.

Albert and Kari showed in [4] that the attractors of IFS (cf. chapter 6) are PWFA computable. This may be understood as the iterated image of a single point set (a point on the attractor of the IFS) under a finite sequence of contractive affine transformations.

If a PWFA X over $\mathcal{K} \subset \mathbb{C}$ meets the strong constraint $S(X) = T(X)$, then the iterated image of $S(X)$ under each finite sequence of (not necessarily contractive) affine transformations is PWFA computable. Examples of this closure are depicted in Figure 5.5. The proof is based on Lemma 5.3, for which we define the following operators.

Definition 5.4 *Let $\mathcal{S}$ be a semiring and $d \in \mathbb{N}^+$. Then we define the operators $\diamond_d : \mathcal{S}^{d\times d} \mapsto \mathcal{S}^{d\times d^3}$ and $\diamond_d^{-1} : \mathcal{S}^{d\times d^3} \mapsto \mathcal{S}^{d\times d}$ as the functions*

$$\diamond_d(A) = \begin{pmatrix} \mathcal{A} & O & \dots & O \\ O & \mathcal{A} & \dots & O \\ & & \vdots & \\ O & O & \dots & \mathcal{A} \end{pmatrix} \tag{5.16}$$

for each $A \in \mathcal{S}^{d\times d}$, where

$$\mathcal{A} = \begin{pmatrix} A(1,1) & A(2,1) & \dots & A(d,1) & A(1,2) & A(2,2) & \dots \end{pmatrix} \tag{5.17}$$

and O is the zero vector of $\mathcal{S}^{d^2}$ and

$$\diamond_d^{-1}(B) = \begin{pmatrix} B(1,p(1,1)) & B(1,p(1,2)) & \dots & B(1,p(1,d)) \\ B(2,p(2,1)) & B(2,p(2,2)) & \dots & B(2,p(2,d)) \\ & \vdots & & \\ B(d,p(d,1)) & B(d,p(d,2)) & \dots & B(d,p(d,d)) \end{pmatrix} \tag{5.18}$$

for $B \in \mathcal{S}^{d\times d^3}$, where $p(i,j) = (i-1)d^2 + (j-1)d + i$. Further we define the operator $\triangle_d : \mathcal{S}^{d\times d} \mapsto \mathcal{S}^{d^3\times d^3}$ as

$$\triangle_d A = \begin{pmatrix} A^T & Q & \dots & Q \\ Q & A^T & \dots & Q \\ & \vdots & & \\ Q & Q & \dots & A^T \end{pmatrix}. \tag{5.19}$$

for $A \in \mathcal{S}^{d\times d}$, where Q is the zero matrix of $\mathcal{S}^{d\times d}$.

Definition 5.5 *Let $\mathcal{S}$ be a semiring and d,n positive natural numbers such that $d \leq n$. Then we define the operators $\diamond_{d,n} : \mathcal{S}^{d\times n} \mapsto \mathcal{S}^{d\times d^3}$ and $\diamond_{d,n}^{-1} : \mathcal{S}^{d\times d^3} \mapsto \mathcal{S}^{d\times n}$ as the functions*

$$\diamond_{d,n}(A) = \diamond_d \begin{pmatrix} A(1,1) & A(1,2) & \dots & A(1,d) \\ & \vdots & & \\ A(d,1) & A(d,2) & \dots & A(d,d) \end{pmatrix} \tag{5.20}$$

for each $A \in \mathcal{S}^{d\times n}$ and

$$\diamond_{d,n}^{-1}(B) = \begin{pmatrix} \diamond_d^{-1}B & O_{d,n-d} \end{pmatrix} \tag{5.21}$$

for each $B \in \mathcal{S}^{d\times d^3}$, where $O_{u,v}$ denotes the zero matrix in $\mathcal{S}^{u\times v}$.

Lemma 5.1 *Let $\mathcal{S}$ be a semiring and $d \in \mathbb{N}^+$. Let $A \in \mathcal{S}^{d\times d}$. Then*

$$\diamond_d^{-1}(\diamond_d(A)) = A. \tag{5.22}$$

Lemma 5.2 *Let $\mathcal{S}$ be a semiring, $d \in \mathbb{N}^+$ and $n \in \mathbb{N}$ such that $n \geq d$. Then there are matrices $P_d \in \mathcal{S}^{d^3\times d}$ and $P_{d,n} \in \mathcal{S}^{d^3\times n}$ such that*

$$\diamond_d^{-1}(B) = BP_d \tag{5.23}$$

for each $B \in \mathcal{S}^{d\times d^3}$ and

$$\diamond_{d,n}^{-1}(B) = BP_{d,n} \tag{5.24}$$

for each $B \in \mathcal{S}^{d\times d^3}$.

Proof: P_d is given by

$$P_d(u,v) = \begin{cases} 1 & \text{if } (u,v) = ((i-1)d^2 + (j-1)d + i, j) \text{ for } i,j \in \{1,\dots,d\} \\ 0 & \text{otherwise,} \end{cases} \tag{5.25}$$

for $u = 1, \ldots, d^3, v = 1, \ldots, d$. $P_{d,n}$ can be defined as

$$P_{d,n} = \begin{pmatrix} P_d & O_{d^3,n-d} \end{pmatrix}, \tag{5.26}$$

where $O_{u,v}$ denotes the zero matrix in $\mathcal{S}^{u\times v}$. □

The matrices P_d and $P_{d,n}$ defined in the proof of Lemma 5.2 are equivalent to the operators $\diamond_d^{-1}$ and $\diamond_{d,n}^{-1}$. In the following we will use the symbols $\diamond_d^{-1}$ and $\diamond_{d,n}^{-1}$ for the matrices P_d and $P_{d,n}$ respectively. It will be clear from the context, in which cases the symbols refer to the operators and and in which they refer to the matrices.

Lemma 5.3 *Let $\mathcal{S}$ be a commutative semiring and $d \in \mathbb{N}^+$. Let $A, B \in \mathcal{S}^{d\times d}$. Then*

$$BA = \diamond_d^{-1}((\diamond_d A)\triangle_d B) = (\diamond_d A)\triangle_d B \diamond_d^{-1} \quad . \tag{5.27}$$

Remark 5.1 *We require the multiplication in $\mathcal{S}$ to be commutative, because the application of the $\diamond_d$ operator as well as the construction of the matrix $\triangle$ effectively transposes A and B, thus we are by some detour computing BA as $(A^T B^T)^T$.*

Theorem 5.11 *Let $(\mathcal{K}^d, m)$ for some $d \in \mathbb{N}^+$ denote a complete metric space, where $\mathcal{K} \subset \mathbb{C}$ is a semiring and the metric m is given as $m(x,y) = |x-y|$ for $x, y \in \mathbb{C}^d$ (i.e. the metric induced by the L^2 or Euclidean norm). Furthermore, let X be a PWFA of dimension d over $\mathcal{K}$ such that $S(X) = T(X)$ and let $g_k(x) = B_k x + c_k$ be affine transformations on $\mathcal{K}^d$ for each $k \in \Gamma = \{1, \ldots, r\}$ for some positive integer r. There is a PWFA Z of dimension d over $\mathcal{K}$ that computes the iterated image of $S(X)$ under $g_1, \ldots, g_r$.*

Proof: Assume without loss of generality that X has exactly one distinct initial state per dimension and that the initial state for dimension k is state k where $I(k,k) = 1$ for each $k = 1, 2, \ldots, d$. Further assume without loss of generality that $\Gamma = \Sigma$. If $|\Gamma| > |\Sigma|$, then we can choose the missing affine transformations as any of those that are already given. If $|\Sigma| > |\Gamma|$, we can introduce new symbols in X while choosing their transition matrices as any of those that are already given. We construct the automaton Z using a new symbol $l_X + 1 \notin \Sigma_X$, thus $\Sigma_Z = \{1, 2, \ldots, l_X, l_X + 1\}$. We call $l_X + 1$ the splitting symbol of Z, because the edges labeled by $l_X + 1$ form a bridge between two otherwise independent sub-automata. Let $h : \Sigma_Z^* \mapsto \Sigma_X^*$ denote the homomorphism that erases $l_X + 1$, i.e. maps $a \in \Sigma_X$ to itself and $l_X + 1$ to ε. Let $w \in \Sigma_Z^*$. Furthermore, let the functions ν be defined as given in Definition 5.3 for the sequence $g_1, \ldots, g_r$. Then we construct Z such that it computes

$$f_Z(w) = \begin{cases} f_X(w) & \text{if } h(w) = w \\ \nu(h(\alpha))(f_X(\beta)) & \text{if } w = \alpha(l_X+1)\beta \text{ for } \alpha \in \Sigma_Z^*, \beta \in \Sigma_X^*. \end{cases} \tag{5.28}$$

If w does not contain the symbol $l_X + 1$, then Z behaves like X. We implement this by integrating X into Z, where the transition matrix for label $l_X + 1$ is chosen to be the zero matrix for the states of X. If the input word contains the symbol $l_X + 1$, then the content of the corresponding states is erased and thus the construction does not interfere with the second case. Now assume that w contains the symbol $l_X + 1$ at least once, i.e. w can be decomposed into $w = \alpha(l_X + 1)\beta$

where $\alpha \in \Sigma_Z^*$ and $\beta \in \Sigma_X^*$ and Z is to compute $f_Z(w) = \nu(h(\alpha))(f_X(\beta))$. Observe that for $u = b_1 \dots b_k \in \Gamma^*$ and $x \in \mathcal{K}^d$ the equation

$$\begin{aligned} \nu(u)(x) &= g_{b_1} \circ g_{b_2} \circ \dots g_{b_k}(x) \\ &= B_{b_k}(\dots(B_{b_2}(B_{b_1}x + c_1) + c_2)\dots) + c_{b_k} \\ &= \underbrace{\left(\prod_{i=1}^{k} B_{b_{k-i+1}}\right) x}_{\text{linear transformation } \nu_l(u)(x)} + \underbrace{\left(\prod_{i=1}^{k-1} B_{b_{k-i+1}}\right) c_{b_1} + \dots + c_{b_k}}_{\text{part } \nu_t(u) \text{ generated by iterated translation}} \end{aligned} \tag{5.29}$$

holds. As $\nu_t(u)$ does not depend on the variable x, we can rewrite the second case of equation 5.28 as

$$f_Z(\alpha(l_X+1)\beta) = \underbrace{\nu_l(\alpha)(f_X(\beta))}_{f_{Z'}(\alpha(l_X+1)\beta)} + \underbrace{\nu_t(\alpha)}_{f_{Z''}(\alpha(l_X+1)\beta)} \tag{5.30}$$

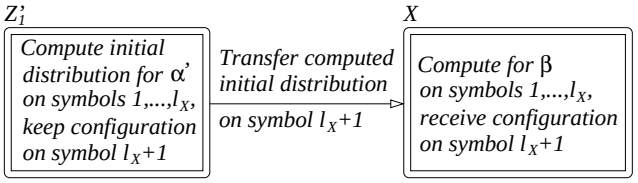
and decompose the corresponding part of Z into the sub-automata Z' and Z'' such that $f_Z(w) = f_{Z'}(w) + f_{Z''}(w)$ for each $w \in \Sigma_Z^*(l_X+1)\Sigma_X^*$. We firstly consider the construction of Z'. Let $\alpha \in \Sigma_Z^*$, $\alpha' = h(\alpha) = a_1 \dots a_p$, $B_u = B_{d_k} \dots B_{d_2} B_{d_1}$ for each $u = d_1 \dots d_k \in \Sigma_X^*$ and $\beta = b_1 \dots b_q \in \Sigma_X^*$. Then the function $f_{Z'}$ computed by Z' has to satisfy

$$f_{Z'}(\alpha(l_X+1)\beta) = f_{\Xi[B_{\alpha'} I_X](X)}(\beta), \tag{5.31}$$

i.e. it has to yield the same value for $\alpha(l_X+1)\beta$ that the automaton $\Xi[B_{\alpha'} I_X](X)$ computes for β. As we have shown in Lemma 5.3, the function of Z' can be implemented as

$$\begin{aligned} f_{Z'}(\alpha(l_X+1)\beta) &= f_{\Xi[B_{\alpha'} I_X](X)}(\beta) \\ &= B_{\alpha'} I_X \textstyle\prod_{i=1}^{q} A_{X_{b_i}} F_X \\ &= \diamond_{d,n}^{-1}(\diamond_{d,n} I_X \triangle_d B_{\alpha'}) \textstyle\prod_{i=1}^{q} A_{X_{b_i}} F_X \\ &= \diamond_{d,n}^{-1}(\diamond_{d,n} I_X \textstyle\prod_{i=1}^{p} \triangle_d B_{a_i}) \textstyle\prod_{i=1}^{q} A_{X_{b_i}} F_X \\ &= \diamond_{d,n} I_X \textstyle\prod_{i=1}^{p} \triangle_d B_{a_i} \diamond_{d,n}^{-1} \textstyle\prod_{i=1}^{q} A_{X_{b_i}} F_X \quad . \end{aligned} \tag{5.32}$$

Thus we give the automaton Z' the following structure:



Starting with $Z' = X$, we add d^3 states to Z' and call the respective sub-automaton containing these additional states Z'_1. Let U_j denote the identity matrix of $\mathcal{K}^{j \times j}$. We set $I_{Z'_1} = \diamond_d U_d$, use the zero vector of $\mathcal{K}^{d^3}$ as $F_{Z'_1}$, and define the transition matrices $A_{Z'_{1a}}$ of Z'_1 by

$$A_{Z'_{1a}} = \begin{cases} \diamond B_a & \text{for } a \in \Sigma_X \\ U_{d^3} & \text{for } a = l_X + 1 \end{cases} \tag{5.33}$$

for $a = 1, \dots, l_X + 1$. The only interaction between the sub-automata Z'_1 and X of Z' happens when Z' reads the symbol $l_X + 1$. Then we transfer the new

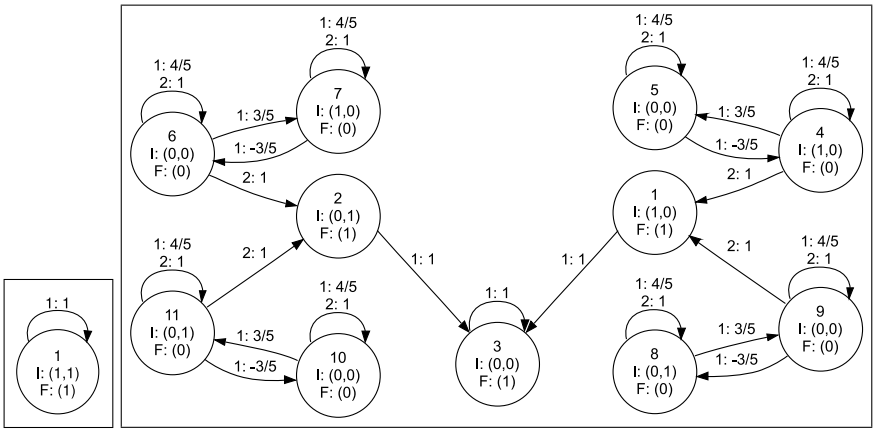


Figure 5.6: The real PWFA X of dimension 2 on the left computes the constant function $f_X \equiv (1\ \ 1)^T$. The automaton Z' shown on the right computes the iterated image of $S(X)$ under the single map $g(x) = Bx$ where B is a rotation by the angle $\mathrm{acos}(0.8)$. Thus $S(Z')$ is a circle of radius $\sqrt{2}$.

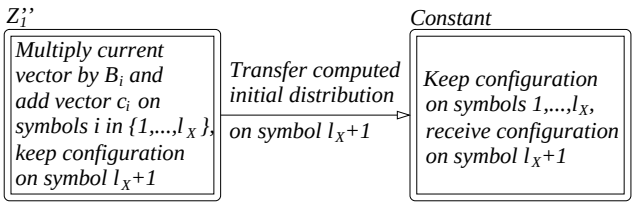
initial distribution computed in Z'_1 into X using the $\diamond^{-1}$ operator, which we can implement as a matrix as shown above. An example of this construction is shown in Figure 5.6. In this example the constant function is subject to a single map based iterated image, where the map is a rotation by an angle of $\mathrm{acos}(0.8)$. The automaton is first transformed into an automaton in initial normal form using the method presented in the proof of Lemma 4.7. After this step it consists of the states 1, 2 and 3, where state 1 is the initial state of the first dimension, state 2 the initial state of the second dimension and state 3 is state 1 of the original automaton. Then we add the states 4 up to 11, which correspond to the sub-automaton Z'_1 and assign the initial matrix $\diamond U_2$ to this group of states. The rotation map is applied to the original initial distribution in these states. Whenever the symbol 2 is consumed, the new initial distribution is transfered into the states 1 and 2. We can thus define Z' as

- $Q_{Z'} = \{1, \ldots, |Q_X| + d^3\}$,
- $\Sigma_{Z'} = \{1, \ldots, l_X + 1\}$,
- $A_{Z'_i} = \begin{cases} \begin{pmatrix} A_{X_i} & \\ & \Delta B_i \end{pmatrix} & \text{for } i = 1, \ldots, l_X \\ \begin{pmatrix} & \\ \diamond_d^{-1} & U_{d^3} \end{pmatrix} & \text{for } i = l_X + 1, \end{cases}$
- $I_{Z'} = \begin{pmatrix} I_X & \diamond_d U_d \end{pmatrix}$ and
- $F_{Z'} = \begin{pmatrix} F_X^T & O^T \end{pmatrix}^T$ where O denotes the zero vector of $\mathcal{K}^{d^3}$.

Let us now consider the automaton Z''. Let $\alpha \in \Sigma_Z^*$, $\alpha' = h(\alpha) = a_1 \dots a_p$ and $\beta \in \Sigma_X^*$. The function $f_{Z''}$ computed by Z'' needs to satisfy

$$\begin{aligned} f_{Z''}(\alpha(l_X+1)\beta) &= \nu_t(\alpha) \\ &= \textstyle\prod_{i=1}^{p-1} B_{a_{p-i+1}} c_{a_1} + \ldots + c_{a_p} \\ &= \left((c_{a_1}^T \textstyle\prod_{i=2}^{p} B_{a_i}^T) + \ldots + c_{a_p}^T \right)^T \\ &= \left(c_{a_p}^T + (\ldots + (c_{a_2}^T + (c_{a_1}^T B_{a_2}^T)) B_{a_3}^T \ldots) \right)^T \quad . \end{aligned} \tag{5.34}$$

When Z'' has consumed α, it no longer changes its configuration, i.e. computes the constant function on some previously computed initial distribution. As all dimensions have this constant function in common, we do not need as many states as in the general $\diamond$ construction above. Z'' implements the following operation scheme:



Let U_k denote the identity matrix of $\mathcal{K}^{k\times k}$ for $k \in \mathbb{N}^+$. Furthermore, let $\mathcal{B}_i \in \mathcal{K}^{(d+1)\times(d+1)}$ be given by

$$\mathcal{B}_i = \begin{pmatrix} B_i^T & \\ c_i^T & U_1 \end{pmatrix} \tag{5.35}$$

for $i = 1, \ldots, m$ and $P \in \mathcal{K}^{d(d+1)\times 1}$ given by

$$P(i,1) = \begin{cases} 1 & \text{if } i = (j-1)(d+1)+j \text{ for } j \in \{1, \ldots d\} \\ 0 & \text{otherwise} \end{cases} \quad . \tag{5.36}$$

Then we define Z'' as

- $Q_{Z''} = \{1, \ldots, d(d+1)+1\}$,
- $\Sigma_{Z''} = \{1, \ldots, l_X+1\}$,
- $A_{Z_i''} = \begin{cases} \begin{pmatrix} U_1 & O_1^T & O_1^T & \ldots & O_1^T \\ O_1 & \mathcal{B}_i & O_2 & \ldots & O_2 \\ O_1 & O_2 & \mathcal{B}_i & \ldots & O_2 \\ & & \vdots & & \\ O_1 & O_2 & O_2 & \ldots & \mathcal{B}_i \end{pmatrix} & \text{for } i = 1, \ldots, l_X \\ \begin{pmatrix} & \\ P & U_{d(d+1)} \end{pmatrix} & \text{for } i = l_X + 1 \end{cases}$

  where O_1 is the zero vector of $\mathcal{K}^{d+1}$ and O_2 the zero matrix in $\mathcal{K}^{(d+1)\times(d+1)}$,

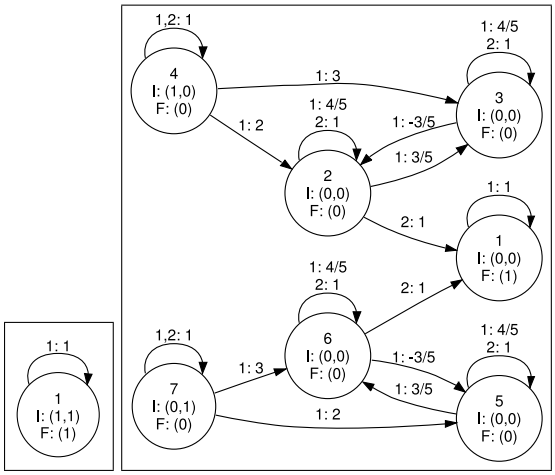


Figure 5.7: The automaton X on the left computes the constant function $f_X = (1\ \ 1)^T$. The automaton Z'' shown on the right computes the translational part of an iterated image based on the single map consisting of a rotation by the angle $\text{acos}(\frac{4}{5})$ followed by a translation by the vector $(2\ \ 3)^T$.

- $I_{Z''}$ is 1 at column $1+i(d+1)$ in row i and zero otherwise for $i = 1,\ldots,d$ and
- $F_{Z''}$ is 1 for state 1 and zero otherwise.

An example of this construction is shown in Figure 5.7. State 1 keeps its configuration for symbol 1 and drops its configuration for symbol 2. The multiplicative part is performed on states $2,3,5$ and 6. States 4 and 7 are the initial states for dimension 1 and 2 respectively. Their value multiplied by the corresponding numbers is added to states $2,3,5$ and 6 to implement the translation. Whenever the symbol 2 is read, the new initial distribution is transfered into state 1.
Z computes the function f_Z given in equation 5.28 by construction. Thus it is clear that

$$\nu(S(X)) \subset S(Z) \tag{5.37}$$

holds. The converse direction

$$S(Z) \subset \nu(S(X)) \tag{5.38}$$

follows because $S(X) = T(X)$, i.e. f_X does not produce any vectors v that have a positive distance $m(v,S(X))$ from the set $S(X)$ and each g_i is continuous for $i = 1,\ldots,m$. □

The constraint $S(X) = T(X)$ required in Theorem 5.11 is strong but met by most practically relevant PWFA. The following example shows that neither the ST-consistency of an automaton nor the invertibility of the applied maps is in general sufficient to guarantee that the construction given in the proof of Theorem 5.11 produces an automaton computing the correct iterated image.

Example 5.1 *Let $X = (\{1\},\{1\},((\frac{1}{2})),(1),(1))$ be a PWFA of dimension 1 over $\mathbb{R}$. Furthermore, let the function $g_1 : \mathbb{R} \mapsto \mathbb{R}$ be given by $g_1(x) = x$ for all $x \in \mathbb{R}$. The*

set computed by X is $S(X) = \{0\}$, X is ST-consistent and g_1 is an invertible affine transformation on $\mathbb{R}$. The construction given in the proof of Theorem 5.11 would however yield a PWFA computing a function that produces each value 2^{-i} infinitely often for $i = 0, 1, \ldots$. Thus the constructed automaton would not compute the iterated image $\nu(S(X)) = \{0\}$ of $S(X)$ under g_1.

In the construction of ν_l above we have chosen to apply the transformations on the left side of the multiplication chain, inserting matrix multiplications between the original initial matrix and the original transition matrices. We could have chosen to apply the transformations on the right side of the multiplication chain as well (i.e. between the original transition matrices and the original final distribution). This however would usually require the generation of d^2 full copies of the the input automaton, while in our case it is sufficient to have d^3 additional states. As the state number is typically larger than the dimension, the construction above is usually a good choice.

In the construction of the matrices $A_{Z'_{l_X+1}}$ and $A_{Z''_{l_X+1}}$ given above, we can substitute the sub-matrices $\diamond_d^{-1}$ and P by $\diamond_d^{-1}B$ and PB for any matrix $B \in \mathcal{K}^{d\times d}$ and thus apply a final transformation to the result of the transformation sequence computed before. If we use multiple split matrices like the transition matrix for the symbol $l_X + 1$, we can also have multiple final transforms.

We state this in the following theorem.

Definition 5.6 *Let $d \in \mathbb{N}^+$. Furthermore, let $\mathcal{S}$ be a semiring, let $g_1, g_2, \ldots, g_r$ be a finite sequence of affine transformations on $\mathcal{S}^d$ and let $L_1, L_2, \ldots, L_k$ be a finite sequence of linear transformations on $\mathcal{S}^d$. Let $S \subset \mathcal{S}^d$ and let $\nu(S)$ denote the iterated image of S under $g_1, \ldots, g_r$. Then we call the set $\bigcup_{i=1}^{k} L_i\nu(S)$ the iterated image of S under $g_1, \ldots, g_r$ transformed by $L_1, L_2, \ldots L_k$.*

Theorem 5.12 *Let $(\mathcal{K}^d, m)$ for some $d \in \mathbb{N}^+$ denote a complete metric space, where $\mathcal{K} \subset \mathbb{C}$ is a semiring and the metric m is given as $m(x,y) = |x-y|$ for $x, y \in \mathbb{C}^d$ (i.e. the metric induced by the L^2 or Euclidean norm). Furthermore, let X be a PWFA of dimension d over $\mathcal{K}$ such that $S(X) = T(X)$, let $g_1, \ldots, g_r$ be a finite sequence of affine transformations on $\mathcal{K}^d$ and let $L_1, \ldots, L_k$ be a finite sequence of linear transformations on $\mathcal{K}^d$. There is a PWFA Z of dimension d over $\mathcal{K}$ that computes the iterated image of $S(X)$ under $g_1, \ldots, g_r$ transformed by $L_1, L_2, \ldots, L_k$.*

The difference between the automaton produced by the construction of Theorem 5.12 and one that is produced by Theorem 5.11 and then subject to the same linear transformations via the constructions given in the proofs of Theorem 5.1 and Theorem 5.5 is only a delay by a finite number of symbols. This may however be relevant when synchronizing the computation of two automata for addition or multiplication. The construction in the proof of Theorem 5.11 can also be performed without previously computing the initial normal form. This circumvents the delay by one symbol introduced by the generation of the initial normal form but will usually produce an automaton that has a larger number of states. Another simple modification lets us compute a kind of Cauchy product.

Definition 5.7 *Let $X = (Q, \Sigma = \{1, \ldots, l\}, W, I, F)$ be a PWFA of dimension d over the semiring $\mathcal{S}$ and let $\mathcal{B} = B_1, \ldots, B_l$ a finite sequence of linear transformations on $\mathcal{S}^d$. Let $\nu : \Sigma^* \mapsto \mathcal{S}^d$ be defined by*

$$\nu(u) = B_{a_k} \ldots B_{a_1} \tag{5.39}$$

for each $u = a_1 \dots a_k \in \Sigma^*$. Then we define the product $(\mathcal{B}f_X)$ of $\mathcal{B}$ and f_X as the function given by

$$(\mathcal{B}f_X)(w) = \sum_{uv=w} \nu(u)(f_X(v)) \tag{5.40}$$

for each $w \in \Sigma^*$.

Theorem 5.13 *Let $X = (Q, \Sigma = \{1, \dots, l\}, W, I, F)$ be a PWFA of dimension d over the semiring $\mathcal{S}$ and $\mathcal{B} = B_1, \dots, B_l$ a finite sequence of linear transformations of $\mathcal{S}^d$. There is a PWFA Z of dimension d over $\mathcal{S}$ that computes the function $f_Z = (\mathcal{B}f_X)$.*

Proof: The automaton Z is obtained from the automaton Z' constructed in the proof of Theorem 5.11 by placing the sub-matrix $\diamond_d^{-1} B_i$ in matrix $A_{Z'_i}$ at the position where $\diamond_d^{-1}$ is located in $A_{Z'_{l_X+1}}$ for $i = 1, \dots, l$ and removing the splitting label from Z'. □

5.1.6 Regular restriction

Let X be a PWFA over some semiring $\mathcal{S}$ with input alphabet Σ_X and $L \subset \Sigma_X^*$. We define the restriction $S(X, L)$ of $S(X)$ under L as the set that would be produced by X, if the definition of $S_k(X)$ given in (4.3) were replaced by $S_k(X) = \{f_X(w) | w \in L \cap \Sigma^k\}$.

Theorem 5.14 *Let X be a PWFA of dimension d over some semiring $\mathcal{S}$ and $L \subset \Sigma_X^*$ regular. Then there is a PWFA Z of dimension d over $\mathcal{S}$ computing the set $S(Z) = S(X, L)$.*

Proof: The proof is divided into two sections, depending on the empty- or non-emptiness of $S(Z)$.

1. Case: $S(Z)$ is empty. Then Z can be chosen to be any PWFA that computes the empty set.

2. Case: $S(Z)$ is not empty. Then there is a point $z_1 \in S(Z)$. As L is regular, there is a deterministic finite automaton B that accepts $L = L(B)$. Assume without loss of generality that B is complete. Z consists of $|Q_B|$ copies of X with one additional state q_{z_1} per copy. The copy that corresponds to the initial state s_B of B has the initial distribution I_X on the original states and z_1 on the additional state. The other copies have vanishing initial distributions. Every copy that corresponds to a state in F_B has the final distribution F_X on the original states and zero on the additional state, every copy for states in $Q_B \setminus F_B$ has a vanishing final distribution on the original states and one on the additional state. The edges in Z are defined as follows:

   (a) If there is an edge in X from state $q_1 \in Q_X$ to $q_2 \in Q_X$ with weight $w \in \mathcal{S}$ for label $l \in \Sigma_X$ and an edge in B from state $p_1 \in Q_B$ to state $p_2 \in Q_B$ for the same label, then there is an edge with weight w in Z from q_1 in the copy that corresponds to p_1 to q_2 in the copy the corresponds to p_2 for the label l:

$$(p_1, l, p_2) \in R_B, (q_1, l, q_2, w) \in R_X \Rightarrow (q_{1_{p_1}}, l, q_{2_{p_2}}, w) \in R_Z \ . \tag{5.41}$$

(b) The edges for the additional states are constructed in the same manner:

$$(p_1, l, p_2) \in R_B \Rightarrow (q_{z1_{p_1}}, l, q_{z1_{p_2}}, 1) \in R_Z \tag{5.42}$$

Z computes the point z_1 on words in $\Sigma_X^* \setminus L$ and the same points as X on the words in L, what completes the proof.

□

The proof of Theorem 5.14 is based on a Cartesian product construction that is usually also applied to prove other theorems in automata theory, e.g. the closure of the set of regular sets under intersection. We will show in section 5.2 that the membership and emptiness problems for PWFA, i.e. "Given a PWFA X and a vector v, is $v \in S(X)$?" and "Given a PWFA X, does $S(X) = \emptyset$ hold?", are recursively undecidable. Thus the construction above is not totally recursive.

5.1.7 Set intersection and complement

We conjecture that the set $\mathcal{D}(\mathbb{C})$ is not closed under set intersection. Some arguments that support this conjecture will be given in the end of subsection 7.2.2. If a PWFA X of dimension d over $\mathbb{C}$ computes a bounded non-empty set $S(X)$, then $S(X)$ is closed and consequently its complement $\mathbb{C}^d \setminus S(X)$ is an open set and thus not computable by a PWFA. If we define the complement of the set computed by a PWFA X of dimension d over $\mathbb{C}$ as the union of the set $\mathbb{C}^d \setminus S(X)$ and the set of border points of $S(X)$, then the closure under set complement would by DeMorgan's rule also imply the closure under set intersection. Thus we conjecture that this kind of closure under the set complement operation also does not hold.

5.2 Decision problems

In this section we will show that some basic problems for PWFA, including the membership, emptiness and equivalence problem, are recursively undecidable. As a consequence of these undecidability results we conclude that there is no state minimization algorithm for general PWFA. These results were first presented in [92]. We will mostly be considering PWFA of dimension 1 over $\mathbb{Z}$ in this section. Thus most of the discussed automata have a strong correlation to integer WFA, which have already been discussed in e.g. [53] and [52]. The vectors in the set computed by an integer PWFA of dimension 1 are exactly those that are produced infinitely often by a certain integer WFA. The proof methods used in this section are similar to some of those used by Eilenberg in e.g. [38]. A proof that is similar to that given for the undecidability of vector membership in a PWFA generated set was used by Paterson in [78], where the decision, whether a system of 3×3 matrices is mortal, was also reduced to Post's correspondence problem [80]. Paterson's work was also used in [51] to show undecidability for some problems in matrix theory. The encoding of Post's correspondence problem used below can also be found in example 1-24 of [72]. Nonetheless we will give a complete description in PWFA form, as the automaton provided in the proof of the undecidability of the membership problem can

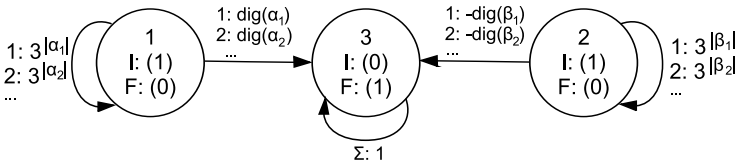


Figure 5.8: PCP automaton.

be extended to an automaton that can be used to show the undecidability of the emptiness problem in a simple way.

5.2.1 Membership problem

Definition 5.8 *We define the membership problem for PWFA as the decision problem "Given a PWFA X of dimension d over some semiring $\mathcal{S}$ and a vector $v \in \mathcal{S}^d$, does $v \in S(X)$ hold?".*

The following theorem shows that the membership problem is recursively undecidable.

Theorem 5.15 *The membership problem for PWFA of dimension 1 over $\mathbb{Z}$ is recursively undecidable.*

Proof: Let $\{(\alpha_1,\beta_1),(\alpha_2,\beta_2),\ldots,(\alpha_k,\beta_k)\}$ denote an arbitrary instance of Post's correspondence problem (PCP, [80]) for $\alpha_i,\beta_i \in \Gamma^+$ and $k \in \mathbb{N}^+$, where $\Gamma = \{1, 2\}$. The choice of the binary alphabet is without loss of generality, as we can encode each instance of PCP over the binary alphabet. Let $\Sigma = \{1,2,\ldots,k\}$. $u = b_1 \ldots b_p \in \Sigma^+$ is said to be a solution of the PCP, if and only if $\alpha_{b_1}\alpha_{b_2}\ldots\alpha_{b_p} = \beta_{b_1}\beta_{b_2}\ldots\beta_{b_p}$. Clearly, if $u \in \Sigma^+$ is a solution, then so is every element of the set u^+, implying that if the PCP has one solution, then it has infinitely many. Let $\mathrm{dig}(w) = \sum_{i=1}^{p} 3^{i-1} b_i$ for each $w = b_1 \ldots b_p \in \Gamma^*$. Now consider the k label 3 state PWFA X shown in Figure 5.8. After reading the word $u = b_1 \ldots b_p \in \Sigma^*$, states 1 and 2 contain the number $3^{\sum_{i=1}^{p}|\alpha_{b_i}|}$ and $3^{\sum_{i=1}^{p}|\beta_{b_i}|}$ respectively. This implies that reading the letter $a \in \Sigma$ after reading $u = b_1 \ldots b_p \in \Sigma^*$ adds the number $\mathrm{dig}(\alpha_a)3^{\sum_{i=1}^{p}|\alpha_{b_i}|} - \mathrm{dig}(\beta_a)3^{\sum_{i=1}^{p}|\beta_{b_i}|}$ to the value contained in state 3. Thus after reading $u = b_1 \ldots b_p \in \Sigma^*$ state 3 has accumulated the value

$$f_X(u) = \mathrm{dig}(\alpha_{b_1}\ldots\alpha_{b_p}) - \mathrm{dig}(\beta_{b_1}\ldots\beta_{b_p}), \tag{5.43}$$

which is 0 for $p > 0$, if and only if u is a valid solution of the PCP. Now assume that the PCP has a solution. Then the automaton produces the value 0 infinitely often as a value in state 3, which is also identical to the function computed by the PWFA. Thus, if the PCP has a solution, then $0 \in S(X)$. If, on the other hand, the PCP has no solution, then the automaton will produce the output 0 only for the empty word. Thus $0 \notin S(X)$. Summarizing this means that $0 \in S(X)$, if and only if the PCP has a solution. As PCP is recursively undecidable, so is the membership problem for PWFA. □

Corollary 5.1 *It is undecidable, whether a given integer WFA assigns 0 to infinitely many words.*

An equivalent statement can be made about the membership in the T set of a PWFA.

Corollary 5.2 *Let X be a PWFA of dimension 1 over $\mathbb{Z}$. It is undecidable whether $0 \in T(X)$.*

Proof: The automaton constructed in the proof of Theorem 5.15 produces 0 for the empty word. We can change this by adding one state that redefines the word function for the empty word to some non-zero value while preserving the original behavior for non-empty words. Let the automaton obtained in this fashion be denoted by X. Then $0 \in T(X)$, if and only if $0 \in S(X)$, which is undecidable according to Theorem 5.15. □

5.2.2 Membership problem for unary alphabet integer PWFA

The following property of unary alphabet integer PWFA given in Theorem 5.16 clearly separates them from integer PWFA using more than one alphabet symbol. The proof is mainly based on results given in [54], which discusses the well-known *Skolem's problem* for linear recurrent sequences. We first provide a simple result on arithmetic progressions.

Definition 5.9 *The arithmetic progressions $b_1 + p_1\mathbb{Z}$ and $b_2 + p_2\mathbb{Z}$ for $b_1, b_2 \in \mathbb{Z}$ and $p_1, p_2 \in \mathbb{Z} \setminus \{0\}$ are called compatible, if their intersection has an infinite number of elements.*

Lemma 5.4 *Let $b_1 + p_1\mathbb{Z}$ and $b_2 + p_2\mathbb{Z}$ for $b_1, b_2 \in \mathbb{Z}$ and $p_1, p_2 \in \mathbb{Z} \setminus \{0\}$ be arithmetic progressions. It is decidable, whether $b_1 + p_1\mathbb{Z}$ and $b_2 + p_2\mathbb{Z}$ are compatible. If the progressions are compatible, then we can effectively construct the progression describing their infinite intersection.*

Proof: We assume without loss of generality that $b_1, b_2 \in \mathbb{N}$, $p_1, p_2 > 0$ and $b_2 \geq b_1$. Let g denote the greatest common divisor of p_1 and p_2. Let $q_1 = \frac{p_1}{g}$ and $q_2 = \frac{p_2}{g}$. Then the arithmetic progressions are compatible, if and only if the linear Diophantine equation

$$(b_2 - b_1) = g\,(m_1 q_1 - m_2 q_2) \tag{5.44}$$

has infinitely many solutions. If g does not divide $(b_2 - b_1)$, then there is no solution. If g divides $(b_2 - b_1)$, then there is at least one solution as an implication of Bézout's identity. Let c be a common member of the progressions and let l be the least common multiple of p_1 and p_2. Then $c + l\mathbb{Z}$ is the infinite intersection of $b_1 + p_1\mathbb{Z}$ and $b_2 + p_2\mathbb{Z}$. □

Definition 5.10 *(Linear recurrent sequence, according to [54]) A sequence $(u_n)_{n=0}^{\infty}$ (or just (u_n) or u_n, for short) is a* linear recurrent sequence*, if it satisfies*

$$u_n = a_{k-1}u_{n-1} + \ldots + a_1 u_{n-k+1} + a_0 u_{n-k} \tag{5.45}$$

for all $n \geq k$ with fixed $a_j \in \mathbb{Z}$ coefficients for $j = 0, \ldots, k-1$. We say that the relation (5.45) is a linear recurrence relation *of* depth *(or* degree*) k. The first k elements $u_0, u_1, \ldots, u_{k-1}$ of the linear recurrent sequence (u_n) in (5.45) are called the* initial conditions*.*

Definition 5.11 *(Skolem's Problem, according to [54]) Given a linear recurrent sequence (u_i), that is, the linear recurrence relation and the initial conditions, determine whether or not there exists $i \geq 0$ such that $u_i = 0$.*

Theorem 5.16 *Let $X = (Q, \{1\}, W, I, F)$ a PWFA of dimension d over $\mathbb{Z}$ and $v \in \mathbb{Z}^d$. It is decidable, whether $v \in S(X)$.*

Proof: We can without loss of generality assume that $v = 0$, because the set of sets computable by unary alphabet PWFA over $\mathbb{Z}$ is effectively closed under translation. According to Lemma 1.1 in [54], the word function of an m-state integer WFA with unary alphabet can be represented in terms of a linear recurrent sequence $u_n = vM^n w$ for $n \geq 1$, where $v^T, w \in \mathbb{Z}^m, M \in \mathbb{Z}^{m \times m}$. According to Theorem 3.5 in the same paper, the set $Z(u_n) = \{i | u_i = 0\}$ is a union of a finite set F and finitely many arithmetic progressions. That means

$$Z(u_n) = F \cup (a_1 + p\mathbb{Z}) \cup (a_2 + p\mathbb{Z}) \cup \ldots (a_r + p\mathbb{Z}). \tag{5.46}$$

The numbers $p, a_1, \ldots, a_r$ can be found algorithmically, especially it is decidable, whether $Z(u_n)$ is infinite. The steps to decide whether $0 \in S(X)$ are the following: Compute, whether every dimension of the PWFA corresponds to a linear recurrent sequence that produces the value 0 infinitely often. If this is not the case, then $0 \notin S(X)$. Now assume that each dimension k of the PWFA represents a linear recurrent sequence u_{k_n} for which $Z(u_{k_n})$ is infinite and let $((p_1, a_{1_1}, \ldots, a_{1_{r_1}}), (p_2, a_{2_1}, \ldots, a_{2_{r_2}}), \ldots)$ be the corresponding parameters of the arithmetic progressions. Clearly, $0 \in S(X)$ if and only if

$$\bigcap_{k=1}^{d} Z(u_{k_n}) = (\ldots (Z(u_{1_n}) \cap Z(u_{2_n})) \cap \ldots) \tag{5.47}$$

is an infinite set. This holds, if and only if there is a compatible family of progressions $(a_{1_{k_1}} + p_1\mathbb{Z}, a_{2_{k_2}} + p_2\mathbb{Z}, \ldots, a_{d_{k_d}} + p_d\mathbb{Z})$ for some choice of $(k_1, k_2, \ldots, k_d) \in \{1, 2, \ldots, r_1\} \times \ldots \times \{1, 2, \ldots, r_d\}$ such that

$$\left(a_{1_{k_1}} + p_1\mathbb{Z}\right) \cap \left(a_{2_{k_2}} + p_2\mathbb{Z}\right) \cap \ldots \cap \left(a_{d_{k_d}} + p_d\mathbb{Z}\right) \tag{5.48}$$

is an infinite set, which we can effectively test as shown above. □

The following theorem establishes a link between unary alphabet PWFA and Skolem's problem.

Theorem 5.17 *Let X be an arbitrary unary alphabet PWFA of dimension 1 over the integers. Assume that there exists an effective algorithm that produces a unary alphabet PWFA Y of dimension 1 over the integers computing the set $S(Y) = T(X)$. Then Skolem's problem is decidable.*

Proof: Let u_n denote a linear recurrent sequence. Then there is a unary alphabet PWFA $X = (Q, \{1\}, (A_1), I, F)$ of dimension 1 that computes the word function $f_X(1^k) = u_k$ for $k > 0$. Without loss of generality let $f_X(\varepsilon) \neq 0$. Now assume that there is an effective algorithm computing a unary alphabet PWFA Y with $S(Y) = T(X)$. Then it is decidable, whether $0 \in S(Y)$ and thus $0 \in T(X)$, which holds, if and only if there is some $k > 0$, for which $u_k = 0$. □

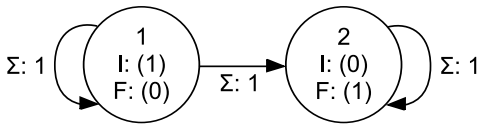


Figure 5.9: Integer PWFA of dimension 1 computing the length of the input word.

5.2.3 Emptiness problem

Theorem 5.18 *Let X be a PWFA of dimension 1 over $\mathbb{Z}$. It is recursively undecidable, whether $S(X) = \emptyset$.*

Proof: Let $\{(\alpha_1,\beta_1),(\alpha_2,\beta_2),\ldots,(\alpha_k,\beta_k)\}$ denote an arbitrary instance of PCP for $\alpha_i,\beta_i \in \Gamma^+$ and $k \in \mathbb{N}^+$, where $\Gamma = \{1,2\}$. Furthermore, let X' be the automaton representing the PCP as given in the proof of Theorem 5.15. We showed that $0 \in S(X')$, if and only if the PCP has a solution. If $S(X')$ does not contain the vector 0, then it could still produce some other vector. So even if the PCP has no solution, $S(X')$ could be non-empty. This can be ruled out by multiplication of X' by the automaton shown in Figure 5.9 (in the sense of the Hadamard product of the word functions) to obtain the PWFA X. On words of finite length $w \in \Sigma^*$ the function computed by X is given by

$$f_X(w) = |w| f_{X'}(w) \quad . \tag{5.49}$$

Remember that X' produces the vector 0 infinitely often, if and only if the PCP it represents has a solution. Thus, if the PCP has a solution, then $0 \in S(X)$, because $|w|0 = 0$ for any $w \in \Sigma^*$ and consequently $S(X) \neq \emptyset$. If, on the other hand, the PCP has no solution, then X' produces the vector 0 only finitely often. This implies that there is some word length $m_0 \in \mathbb{N}$, for which $|f_X(w)| \geq |w|$ for all w of length at least m_0 and there is no sequence $v_1, v_2, \ldots$ of words of increasing length in Σ^* such that $|\bigcup_{i=1}^{\infty}\{f_X(v_i)\}| = 1$. Thus $S(X) = \emptyset$. Summarizing, $S(X) \neq \emptyset$, if and only if the PCP has a solution, which is recursively undecidable. □

Corollary 5.3 *It is undecidable, whether a given integer WFA produces any integer infinitely often.*

5.2.4 Equivalence problem

Theorem 5.19 *Let X and Y be PWFA of dimension 1 over $\mathbb{Z}$. It is recursively undecidable, whether X and Y are equivalent.*

Proof: Assume that equivalence were decidable. Then we could decide the emptiness of $S(Y)$ by choosing X as a PWFA computing the empty set, e.g. the automaton shown in Figure 5.9, which is a contradiction. □

5.2.5 Incomputability of minimal automata for PWFA

Definition 5.12 *Let X be a PWFA of dimension d over the semiring $\mathcal{S}$. We call X (state) minimal, if for all PWFA X' of dimension d over $\mathcal{S}$ computing $S(X') = S(X)$ the equation $|Q_X| \leq |Q_{X'}|$ holds.*

For the rest of the subsection, we will discuss PWFA over $\mathbb{Q}$ exclusively. The rational numbers form a field and there is an effective state minimization for WFA over fields concerning the word function (given that the field operations are effectively computable and order is decidable between the field elements).

Theorem 5.20 *There is no effective state minimization algorithm for PWFA over* $\mathbb{Q}$.

Proof: A one state PWFA over $\mathbb{Q}$ computes the empty set, if and only if the absolute value of each edge weight exceeds one and the product of its initial matrix and final vector is not zero. This implies that emptiness is decidable for one state PWFA over $\mathbb{Q}$. Thus an effective state minimization algorithm would yield an effective decision algorithm for the emptiness problem, which is a contradiction. □

It is an open problem, whether there exists a state minimization algorithm for PWFA concerning the word function the automaton computes. The well-known minimization algorithms for WFA over fields (see e.g. [10, 36, 28]) cannot be applied, because the WFA that can be defined to reflect the word function of a PWFA is not acting on a field as soon as the automaton has a dimension greater than 1 (it is lacking some multiplicative inverses). Eisner shows in [39] that for deterministic WFA over $\mathbb{R}^2$ with component-wise product there is in general no unique minimal automaton. We may still be able to reduce the number of states in a PWFA by applying parts of the WFA minimization algorithm. Considering the algorithm given in [28], we see that it computes the linear spaces generated by the iterated application of the transition matrices to the initial and final distribution. These spaces have a certain dimension that is bounded by the number of states of the automaton. If the dimension of the linear space generated by the initial or final distribution is smaller than the number of states, then the automaton is not minimal and states can be removed. In the PWFA case we have multiple initial distributions and the linear spaces generated by the different initial distributions in general do not match. The final distribution however is unique, so we can reduce the number of the states in a PWFA to the dimension of the linear space generated by the final distribution.

Chapter 6

Fractals

We call an object *fractal*, if it is self similar, i.e. it is exactly or approximately part of itself. The original definition of the the term *fractal set* by Mandelbrot as quoted in [60] is *a set with non-integral Hausdorff dimension*. There are many ways to generate fractal sets ranging from Lindenmayer-systems (or shorter L-systems) over cellular automata to mutually recurrent function systems (cf. [18, 95,22,25,16,17,83,23]). In this chapter we will consider fractal sets that can be produced by PWFA. The construction of PWFA over $\mathbb{R}$ computing attractors of affine hyperbolic iterated functions systems (IFS, cf. [8]) was presented in [4]. In the same paper it was also stated that PWFA over $\mathbb{R}$ are able to compute attractors of recurrent iterated function systems (RIFS, cf. [9]) and mutually recursive function systems (MRFS, cf. [17]).

6.1 Iterated function systems

We first provide some necessary definitions and basic results of IFS theory, both can be found analogously in [8].

Definition 6.1 *A subset $S \subset X$ of a metric space (X,m) is called compact, if every sequence $(x_n)_{n=1}^{\infty}$ in S contains a convergent subsequence having a limit in S.*

Definition 6.2 *(Cauchy sequence) A sequence of points $(x_n)_{n=1}^{\infty}$ in a metric space (X,m) is called a* Cauchy sequence*, if for any given real number $\varepsilon > 0$ there is an integer $N > 0$ such that*

$$m(x_n, x_m) < \varepsilon \tag{6.1}$$

for all $n,m > N$.

Definition 6.3 *(Complete metric space) A metric space (X,m) is called complete, if every Cauchy sequence in (X,m) converges to a point in X.*

Definition 6.4 *Let (X,m) be a complete metric space. Then we define $\mathcal{H}(X)$ as the set containing all non-empty compact subsets of X.*

Definition 6.5 *Let (X,m) be a complete metric space. The* Hausdorff metric *$h(m)$ is defined as*

$$h(m)(A,B) = \max\{\max_{a\in A}\min_{b\in B} m(a,b), \max_{b\in B}\min_{a\in A} m(a,b)\} \tag{6.2}$$

for compact sets $A, B \subset X$.

Theorem 6.1 *Let (X,m) be a complete metric space. Then $(\mathcal{H}(X), h(m))$ is a complete metric space.*

Definition 6.6 *(Contraction mapping) A transformation $f : X \mapsto X$ on a metric space (X,m) is called* contractive *or a* contraction mapping*, if there is a real constant $0 \leq s < 1$ such that*

$$d(f(x), f(y)) \leq sd(x,y) \tag{6.3}$$

for all $x, y \in X$. Any such number s is called a contractivity factor *for f.*

Definition 6.7 *Let $f : X \mapsto X$ be a mapping on a set X. We define the $n-$times application f^n of f recursively as*

$$f^n(x) = \begin{cases} x & \text{if } n = 0 \\ f^{n-1}(f(x)) & \text{if } n > 0 \end{cases} \tag{6.4}$$

for each $x \in X$ and any integer $n \geq 0$.

Theorem 6.2 *(Contraction mapping theorem) Let $f : X \mapsto X$ be a contraction mapping on a complete metric space (X,m). Then f possesses exactly one fixed point $x_f \in X$ and moreover for any point $x \in X$, the sequence $(f^n(x))_{n=0}^{\infty}$ converges to x_f. That is,*

$$\lim_{n \to \infty} f^n(x) = x_f \tag{6.5}$$

for each $x \in X$.

Lemma 6.1 *Let $w : X \mapsto X$ be a contraction mapping on the complete metric space (X,m). Then w is continuous.*

Lemma 6.2 *Let $w : X \mapsto X$ be a continuous mapping on the complete metric space (X,m). Then w maps $\mathcal{H}(X)$ into itself.*

Lemma 6.3 *Let $w : X \mapsto X$ be a contraction mapping on the complete metric space (X,m) with contractivity factor s. Then $w : \mathcal{H}(X) \mapsto \mathcal{H}(X)$ defined by*

$$w(B) = \{w(x) : x \in B\} \tag{6.6}$$

for all $B \in \mathcal{H}(X)$ is a contraction mapping on $(\mathcal{H}(X), h(m))$ with contractivity factor s.

Lemma 6.4 *(Hutchinson operator) Let (X,m) be a complete metric space and n some positive natural number. Let w_k be contraction mappings on $(\mathcal{H}(X), h(m))$ for $k = 1, \ldots, n$. Let the contractivity factor for w_k be denoted by s_k for $k = 1, \ldots, n$. The Hutchinson operator $W : \mathcal{H}(X) \mapsto \mathcal{H}(X)$ over $w_1, \ldots, w_n$ is defined by*

$$\begin{array}{rcl} W(B) & = & w_1(B) \cup w_2(B) \cup \ldots \cup w_n(B) \\ & = & \bigcup_{k=1}^{n} w_k(B) \end{array} \tag{6.7}$$

for each $B \in \mathcal{H}(X)$. Then W is a contraction mapping with contractivity factor $s = \max\{s_1, \ldots, s_n\}$.

Definition 6.8 *A (hyperbolic) iterated function system (IFS) consists of a complete metric space (X,m) together with a finite set of contraction mappings $w_k : X \mapsto X, k = 1,\ldots,n$, with respective contractivity factors s_k, for $k = 1,\ldots,n$. We denote an IFS by writing an $n+2$-tuple $(X,m,w_1,\ldots,w_n)$.*

Theorem 6.3 *Let $(X,m,w_1,\ldots,w_n)$ denote an IFS and let W be the Hutchinson operator over $w_1,\ldots,w_n$. The IFS has a unique fixed point A that is given by*

$$A = \lim_{k\to\infty} W^k(B) \tag{6.8}$$

for any $B \in \mathcal{H}(X)$.

The unique fixed point A of an IFS is called the *attractor* of the IFS.

We will consider the relationship of affine IFS and PWFA.

Definition 6.9 *Let $(\mathcal{S}^d,m)$ for some $d \in \mathbb{N}^+$ be a complete metric space such that $\mathcal{S}$ is a semiring. We call the IFS $(\mathcal{S}^d,m,w_1,\ldots,w_n)$ affine, if each w_k for $k = 1,\ldots,n$ is an affine transformation on $\mathcal{S}^d$.*

The following two algorithms for the approximation of the attractor $\mathcal{A}$ of an IFS $(X,m,w_1,\ldots,w_n)$ are well know.

- The deterministic algorithm: choose an arbitrary element $X_0 \in \mathcal{H}(X)$ and repeatedly apply the Hutchinson operator over $w_1,\ldots,w_n$, i.e. compute $X_k = W^k(X_0)$ for some natural number k. The sequence of sets (X_k) converges to $\mathcal{A}$ for increasing values of k, where the distance between sets is measured using the Hausdorff metric.
- The chaos game: Choose an arbitrary point $x_0 \in X$. Compute points $x_k = w_{j_k}(x_{k-1})$ for $k > 0$, while randomly choosing the elements of the sequence (j_k) from the set $\{1,\ldots,n\}$. Then the accumulation points of the sequence (x_k) are elements of $\mathcal{A}$ and the topological closure of the set of accumulation points equals $\mathcal{A}$ with probability 1.

If the initial set X_0 is large enough (i.e. it contains the attractor of the IFS), then the sets computed by the deterministic algorithm approach the attractor of the IFS from the outside. The Hutchinson operator maps the attractor of the IFS to itself, thus X_k contains $\mathcal{A}$ for each k, but for increasing k the points outside the attractor disappear. If on the other hand X_0 is a subset of $\mathcal{A}$, then the sets X_k approach $\mathcal{A}$ from the inside, i.e. each X_k is a subset of $\mathcal{A}$.

The following result was given in [4] for $\mathcal{S} = \mathbb{R}$.

Theorem 6.4 *Let $(\mathcal{S}^d,m)$ be a complete metric space for some $d \in \mathbb{N}^+$, where $\mathcal{S}$ is a semiring. Furthermore, let $(\mathcal{S}^d,m,w_1,\ldots,w_n)$ be an affine IFS with attractor $\mathcal{A}$. Then there exists a PWFA Z of dimension d over $\mathcal{S}$ such that $S(Z) = \mathcal{A}$.*

Proof: The automaton Z has $d+1$ states and n alphabet symbols. State $d+1$ is the only final state and it has final weight 1. The only initial state of dimension d is state d with initial weight 1. The transition matrices of the automaton are the transformations w_k written in homogeneous coordinates for each $k = 1,\ldots,n$, e.g. for $d = 2$ this means that the transformation

$$w_k(x) = \begin{pmatrix} a & b \\ c & d \end{pmatrix} x + \begin{pmatrix} e \\ f \end{pmatrix} \tag{6.9}$$

is represented by the matrix

$$A_k = \begin{pmatrix} a & b & e \\ c & d & f \\ 0 & 0 & 1 \end{pmatrix}. \tag{6.10}$$

The computation of the PWFA is analogous to the application of the deterministic algorithm. We start with the set $X_0 = \{0\} = S_0(Z) = \{f_Z(\varepsilon)\}$. Thereupon for each positive natural number i the equation

$$X_i = S_i(Z) \tag{6.11}$$

holds. The contraction mapping theorem states that $(X_i)_{i=0}^{\infty}$ is a Cauchy sequence in the complete metric space $(\mathcal{H}(\mathcal{S}^d), h(m))$. As $\mathcal{S}^d$ is complete, (X_i) converges to some set X with equals the attractor $\mathcal{A}$ of the IFS. This convergence implies that for each $a \in \mathcal{A}$ and each real number $\varepsilon > 0$ there is some i_0 such that each $X_i = S_i$ for $i \geq i_0$ contains at least one vector v such that $m(v,a) < \varepsilon$. This in turn means that for each $a \in \mathcal{A}$ and each real $\varepsilon > 0$ there is some i_0 such that there is some word $w_i \in \Sigma^i$ for which $m(f_Z(w_i), a) < \varepsilon$, i.e. there is a sequence of words $w_1, w_2, \ldots$ of increasing length such that $(f_Z(w_i))_{i=1}^{\infty})$ converges to a and thus $a \in S(Z)$. As $\lim_{i\to\infty} h(m)(S_i(Z), \mathcal{A}) = 0$, $S(Z)$ does not contain any points outside $\mathcal{A}$. □

Example 6.1 *The automaton shown in Figure 6.1 computes the well-known fern (cf. [8]). The image it computes is shown in Figure 6.2.*

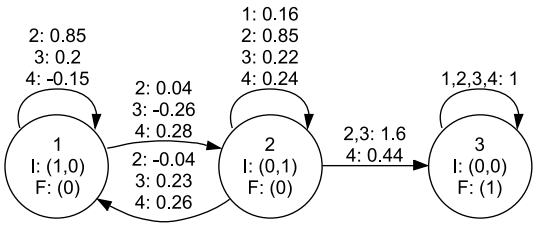


Figure 6.1: Real PWFA of dimension 2 computing the well-known fern.

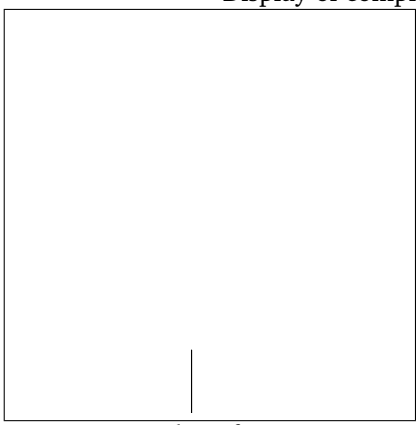
For the rest of the section we will consider complex PWFA exclusively and use the translation invariant metric

$$m(x,y) = |x-y| = \sqrt{\sum_{i=1}^{n} |x(i) - y(i)|^2} \tag{6.12}$$

for $x, y \in \mathbb{C}^n, n \in \mathbb{N}^+$ (i.e. the metric induced by the L^2 or Euclidean norm). Furthermore, we will limit the scope of the discussion by assuming that all appearing weights are component rational, as it avoids undecidability and uncomputability of basic problems and relations.

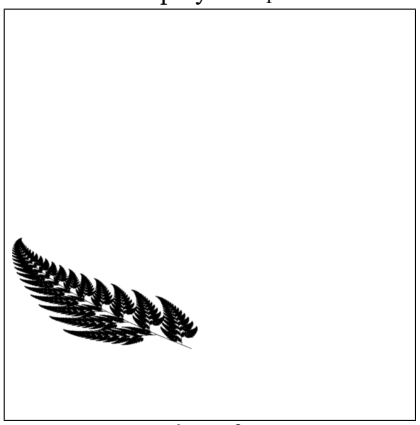
Definition 6.10 *We call a complex number c component rational, if $c = a + ib$ for some $a, b \in \mathbb{Q}$. Likewise we say that a complex vector or complex matrix is*

Display of complete fern attractor $\mathcal{A}$



Display of $w_1\mathcal{A}$

Display of $w_2\mathcal{A}$



Display of $w_3\mathcal{A}$

Display of $w_4\mathcal{A}$

Figure 6.2: The well-known fractal fern. The complete attractor is depicted on top. The four pictures below the complete fern show the images that the attractor is mapped to by the four affine transformations $w_1, \ldots, w_4$. The image $w_1\mathcal{A}$ has been subject to a threshold operation to improve visibility.

component rational, if all elements of the vector or matrix are component rational respectively. We call an affine transformation g of a complex vector space given by $g(x) = Bx + c$ component rational, if the matrix B and the vector c are component rational.

Definition 6.11 *Let $X = (Q, \Sigma, (A_1, \ldots, A_l), I, F)$ be a PWFA of dimension d over $\mathbb{C}$ where all functions $w_{A_k}(x) = A_k x$ for $k = 1, 2, \ldots, l$ are contraction mappings on $\mathbb{C}^n$. Then we say that X is* based on the contraction mappings $w_{A_1}, \ldots, w_{A_l}$ *or shorter X is* based on contraction mappings. *We denote the class of PWFA that are based on contraction mappings by* cPWFA.

We will now show that it is decidable, whether a PWFA over $\mathbb{C}$ using only component rational transition matrices is a member of cPWFA. The proof is based on the following Theorem 6.5 (see [94] for a proof of this result).

Theorem 6.5 *Let $p(x) = \sum_{i=0}^{n} c_i x^i$ denote a real polynomial, where all roots of p are real numbers and $c_i \in \mathbb{Q}$ for $i = 0, \ldots, n$. Furthermore, let $d \in \mathbb{Q}$. It is decidable, whether all roots of p are smaller than d.*

Theorem 6.6 *Let $f(x) = Ax + b$ be an affine transformation on $\mathbb{C}^n$ equipped with the standard metric $m(x, y) = |x - y|$ as defined in (6.12), where A is component rational. It is decidable, whether f is a contraction mapping.*

Proof: f is a contraction mapping with contractivity factor $s < 1$, if and only if

$$|f(x) - f(y)| \leq s|x - y| \tag{6.13}$$

for all $x, y \in \mathbb{C}^n$. Firstly, we note that the additive part of f can be neglected, as

$$|f(x) - f(y)| = |(Ax + b) - (Ay + b)| = |A(x - y)| \ . \tag{6.14}$$

So we can equivalently formulate that f is a contraction mapping with contractivity factor $s < 1$, if and only if

$$|Az| \leq s|z| \tag{6.15}$$

for all $z \in \mathbb{C}^n$. This is trivially true for $z = 0$. On the assumption that $z \neq 0$ we can further rewrite the constraint as

$$\max_{z \neq 0} \frac{|Az|}{|z|} = |A|_2 \leq s < 1 \ . \tag{6.16}$$

The left hand side of (6.16), which is the spectral norm $|A|_2$ of A, equals the square root of the maximal eigenvalue of the real symmetric positive-semidefinite matrix $A^H A$ denoted by A_S, where A^H is the conjugate transpose of A (cf. [59]). According to the spectral theorem A_S can be diagonalized, i.e. the characteristic polynomial $\chi(x)$ of A_S can be decomposed into linear factors such that $\chi(x) = \prod_{i=1}^{n} (x - \lambda_i)$, where the λ_i are eigenvalues of A_S, which are all non-negative real numbers. As χ contains all the primitive factors of the minimal polynomial of A_S, each eigenvalue of A_S is a root of χ. As A is component rational, all numbers in A_S are rational and thus all coefficients of χ_{A_S} are rational. Thus it is decidable, whether all roots of χ_{A_S} are smaller than 1 according to Theorem 6.5. □

The decidability of the contractivity of an affine transformation $f(x) = Ax + b$ where A is component rational can also easily be shown for the metrics inducing the $p = 1$ and $p = \infty$ operator norms. In fact in these cases it is simple to extend the proof to the more general algebraic numbers.

Theorem 6.6 discusses the general case of contraction mappings on $\mathbb{C}^n$, where the proof above shows that the translational part of an affine transformation has no effect on the contractivity of the transformation. The simulation of IFS by PWFA given in [4] uses homogeneous coordinates to simulate the translation of an affine transformation by a matrix multiplication. Each affine transformation w on $\mathbb{R}^d$ for $d \in \mathbb{N}^+$ is represented by a matrix $W \in \mathbb{R}^{(d+1)\times(d+1)}$, where row $d+1$ of W has only zero elements except for the element $W(d+1,d+1)$, which is 1. Thus the matrix W leaves the component $d+1$ of each vector multiplied by W constant. Let the set $\mathcal{C}[d,i](c)$ be defined by

$$\mathcal{C}[d,i](c) = \{(c_1 \ \ldots \ c_{d+1})^T | c_j \in \mathbb{C} \text{ for } j \in \{1,\ldots,d+1\} \setminus \{i\}, c_i = c\} \qquad (6.17)$$

for each $c \in \mathbb{C}$, each $d \in \mathbb{N}$ and each $i \in \{1,\ldots,d+1\}$. If the affine transformation w on $\mathbb{C}^d$ for $d \in \mathbb{N}^+$ is a contraction mapping, then the matrix $W \in \mathbb{C}^{(d+1)\times(d+1)}$ representing w in homogeneous coordinates is a contraction mapping on the set of vectors $\mathcal{C}[d,d+1](c)$ for each $c \in \mathbb{C}$. Vice versa, if there is some $i \in \{1,\ldots,d+1\}$ for a matrix $W \in \mathbb{C}^{(d+1)\times(d+1)}$ for some $d \in \mathbb{N}^+$ such that row i of W has only zero elements except for $W(i,i) = 1$, then the endomorphism on $\mathbb{C}^{d+1}$ described by W is a contraction mapping on $\mathcal{C}[d,i](c)$ for each $c \in \mathbb{C}$, if the endomorphism on $\mathbb{C}^d$ represented by the complex $d \times d$ matrix obtained from W by removing row i and column i is a contraction mapping on $\mathbb{C}^d$. We formulate this in the following lemma.

Lemma 6.5 *Let $f(x) = Ax$ be a linear transformation on $\mathbb{C}^{(d+1)\times(d+1)}$ for some $d \in \mathbb{N}^+$, where row i of A has value 1 in column i and is zero otherwise for some $i \in \{1,\ldots,d+1\}$. Then f is a contraction mapping on $\mathcal{C}[d,i](c)$ for each $c \in \mathbb{C}$, if and only if $f'(x) = A'x$ is a contraction mapping on $\mathbb{C}^d$, where A' is obtained by removing row i and column i from A.*

If we consider a set of matrices $A_1,\ldots,A_k \in \mathbb{C}^{d+1}$ for some $d \in \mathbb{N}^+$ defining the linear functions $f_j(x) = A_j x$ for $j = 1,\ldots,k$ and there exists some $i \in \{1,\ldots,d+1\}$ such that f_j is a contraction mapping on $\mathcal{C}[d,i](c)$ for each $c \in \mathbb{C}$ and each $j \in \{1,\ldots,k\}$, then the function f defined by $f(S) = f_1(S) \cup \ldots \cup f_k(S)$ for each $S \in \mathcal{H}(\mathcal{C}[d,i](c))$ is a contraction mapping on $\mathcal{H}(\mathcal{C}[d,i](c))$ for each $c \in \mathbb{C}$. This means that the result given in Lemma 6.5 can be generalized to multiple affine transformations represented using homogeneous coordinates.

As the contraction mapping property is decidable for component rational affine mappings, we can state the following corollary from the theorem given above.

Corollary 6.1 *Let X be a PWFA of dimension d over $\mathbb{C}$ where the transition matrices of X are all component rational. It is decidable, whether X is a member of the class cPWFA.*

The sets computed by automata in cPWFA can be described in terms of IFS.

Theorem 6.7 *Let $X = (\{1,2,\ldots,n\},\Sigma,(A_1,\ldots,A_l),I,F)$ be a PWFA of dimension d over $\mathbb{C}$ based on the contraction mappings $w_{A_1},\ldots,w_{A_l}$ and let W the Hutchinson operator over $w_{A_1},\ldots w_{A_l}$ acting on $\mathcal{H}(\mathbb{C}^n)$. Then $S(X) = I\lim_{i\to\infty} W^i(\{F\})$ and $I\lim_{i\to\infty} W^i(\{F\}) = \lim_{i\to\infty} IW^i(\{F\})$.*

Proof: For $i \in \mathbb{N}$ we have

$$\begin{aligned} S_i(X) &= \left\{ f_X(w) | w \in \Sigma^i \right\} \\ &= \left\{ I \prod_{j=1}^{i} A_{a_j} F | w = a_1 \ldots a_i \in \Sigma^i \right\} \\ &= IW^i(\{F\}) \ . \end{aligned} \tag{6.18}$$

Thus we have to proof the equation

$$\bigcap_{j=0}^{\infty} \overline{\bigcup_{i=j}^{\infty} IW^i(\{F\})} = I \lim_{j\to\infty} W^j(\{F\}) \ . \tag{6.19}$$

As W is a contraction mapping on a complete metric space, it has a unique fixed point and the limit $\lim_{j\to\infty} W^j(\{F\})$ exists as a compact (and therefore closed) set in $\mathcal{H}(\mathbb{C}^n)$. Thus we have $I \lim_{j\to\infty} W^j(\{F\}) = \lim_{j\to\infty} IW^j(\{F\})$ and can write

$$\begin{aligned} \bigcap_{j=0}^{\infty} \overline{\bigcup_{i=j}^{\infty} IW^i(\{F\})} &= \lim_{j\to\infty} \overline{\bigcup_{i=j}^{\infty} IW^i(\{F\})} \\ &= \overline{\lim_{i\to\infty} IW^i(\{F\})} \\ &= \lim_{i\to\infty} IW^i(\{F\}) \\ &= I \lim_{i\to\infty} W^i(\{F\}) \ . \end{aligned} \tag{6.20}$$

□

Corollary 6.2 *Let $X = (Q, \Sigma, W, I, F)$ be a PWFA of dimension d over $\mathbb{C}$ in cPWFA. $S(X)$ is a linear transformation of the attractor of an IFS of dimension $|Q|$ over $\mathbb{C}$.*

Corollary 6.3 *Let $(\mathbb{C}^k, m, w_1, \ldots, w_n)$ denote an affine IFS with attractor $\mathcal{A}$ for some positive integer k and let $g : \mathbb{C}^k \mapsto \mathbb{C}^d$ be an affine transformation for some positive integer d. There is a PWFA Z in cPWFA that computes $S(Z) = g(\mathcal{A})$.*

Corollary 6.4 *The set of cPWFA computable sets is closed under arbitrary affine transformation for each dimension d.*

Corollary 6.5 *Let $X = (\{1, \ldots, n\}, \Sigma, (A_1, \ldots, A_l), I, F)$ be a PWFA of dimension d over $\mathbb{C}$ based on the contraction mappings $w_{A_1}, \ldots, w_{A_l}$ and let W the Hutchinson operator over $w_{A_1}, \ldots, w_{A_l}$ acting on $\mathcal{H}(\mathbb{C}^n)$. Let $\mathcal{A}$ denote the attractor of the IFS given by W. If $F \in \mathcal{A}$, then $S(X) = T(X)$.*

Corollary 6.6 *Let X be a PWFA of dimension d over $\mathbb{C}$ computing a set which is not compact. Then at least one transition matrix of X does not describe a contraction mapping.*

The set computed by an automaton in cPWFA is never empty. Thus the set of PWFA based on component rational contractive transition matrices forms a subset of the PWFA, for which emptiness is decidable.

Concerning the membership problem for automata X in cPWFA, i.e. the problem "Given a PWFA X and a vector v, does $v \in S(X)$ hold?", we conjecture that it is undecidable, but at least in the case of automata over $\mathbb{R}$ using rational coefficients it is likely that the set of rational vectors contained in the complement set $\mathbb{R}^{d_X} \setminus S(X)$ is recursively enumerable. Lawlor and Hart show in [67] that the attractor of an affine IFS can be bounded by means of convex optimization. This kind of bounding approach is not sufficient to proof what is inside

$S(X)$, but we conjecture that the method or an extension of the method is sufficient to show what is outside. An intriguing point in this discussion is that as soon as we can proof membership for only a single point p of the attractor of an IFS, we can effectively compute an infinite number of points on the attractor by iterated application of the maps of the IFS (assuming there is an infinite number of points on the attractor). The set $\{p\}$ is compact and non-empty, thus the contraction mapping theorem guarantees that the iterated application of the Hutchinson operator of the IFS lets us approach the attractor of the IFS.

6.2 Recurrent iterated function systems

Recurrent iterated function systems (RIFS) are an extension of IFS that was introduced in [9]. An IFS $(X,m,w_1,\ldots,w_n)$ on a non-empty compact metric space (X,m) is augmented by an n state (finite) recurrent Markov chain.

A Markov chain is given by a number of states n, a row stochastic matrix $P \in [0,1]^{n\times n}$ and an initial distribution $p = (p_1 \ \ldots \ p_n)^T \in [0,1]^n$ such that $\sum_{i=1}^n p_i = 1$. A row stochastic matrix is one in which each element is taken from the interval $[0,1]$ and where the elements of each row sum up to 1. Thus we can understand the element $P(i,j)$ of the matrix P as the probability of going from state j to state i. The Markov chain is recurrent, if the matrix P is irreducible, i.e. for every pair of states (i,j) there is some finite sequence of states $i_1,\ldots,i_k$ such that $P(i_k,i_{k-1})\ldots P(i_2,i_1) \neq 0, i_1 = j, i_k = i$ (cf. [9]). We will only consider initial distributions where only a single element does not vanish and call this single element the initial state of the Markov chain.

A Markov chain can be represented as a deterministic real weighted finite automaton Y with n states and n labels. We choose the transition matrix A_k of Y as

$$A_k(i,j) = \begin{cases} P(i,j) & \text{if } i = k \\ 0 & \text{otherwise} \end{cases} \tag{6.21}$$

for $(i,j,k) \in \{1,\ldots,n\}^3$, i.e. each transition for label k starts at state k with a weight taken from the matrix P. Each state of Y is an initial state with initial weight 1. The final distribution of Y equals the initial distribution of the Markov chain. Then $f_Y(w)$ equals the probability assigned to the path $kw_1w_2\ldots w_j$ by the Markov chain for each word $w = w_j\ldots w_1$, if k is the single initial state of the chain.

Example 6.2 *The automaton shown in Figure 6.3 represents a Markov chain, where the chain starts in state* 1*. Note that some state transitions have probability zero, i.e. are impossible. In this example these are exactly the transitions from each state s to itself.*

We denote an RIFS by a tuple $(X,m,w_1,\ldots,w_n,P)$ where $(X,m,w_1,\ldots,w_n)$ is an IFS and $P \in [0,1]^{n\times n}$ is a row stochastic irreducible matrix. In analogy to IFS we can define a recurrent Hutchinson operator for RIFS and based on the recurrent Hutchinson operator, we can define the attractor of an RIFS (cf. [9, 55, 56]).

Definition 6.12 *Let (X,m) denote a compact metric space and n a positive natural number. We define the function $h^n(m) : \mathcal{H}(X)^n \times \mathcal{H}(X)^n \mapsto \mathbb{R}$ by*

$$h^n(m)(A,B) = \max_{i=1,\ldots,n} h(m)(A_i,B_i) \tag{6.22}$$

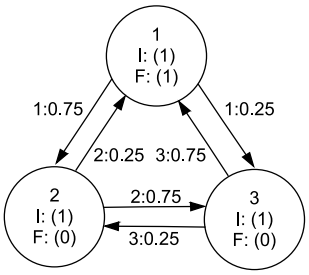


Figure 6.3: Deterministic real WFA representing a Markov chain.

for $A = (A_1, \ldots, A_n), B = (B_1, \ldots, B_n) \in \mathcal{H}(X)^n$.

Lemma 6.6 *Let* (X,m) *denote a compact metric space and* n *a positive natural number. Then* $(\mathcal{H}(X)^n, h^n(m))$ *is a compact metric space.*

Definition 6.13 *Let* $(X,m,w_1,\ldots,w_n,P)$ *be an RIFS. Then the recurrent Hutchinson operator* $w^n : \mathcal{H}(X)^n \mapsto \mathcal{H}(X)^n$ *over* $(w_1,\ldots,w_n,P)$ *is defined by*

$$w^n(A) = (w_1(A), \ldots, w_n(A)) \tag{6.23}$$

for each $A = (A_1, \ldots, A_n) \in \mathcal{H}(X)^n$*, where*

$$w_j(A) = \bigcup_{i:P(i,j)\neq 0} w_i(A_j) \tag{6.24}$$

for each $A = (A_1, \ldots, A_n) \in \mathcal{H}(X)^n$ *and* $j = 1, \ldots, n$.

Theorem 6.8 *Let* $(X,m,w_1,\ldots,w_n,P)$ *be an RIFS. Then the recurrent Hutchinson operator* $w^n : \mathcal{H}(X)^n \mapsto \mathcal{H}(X)^n$ *over* $(w_1,\ldots,w_n,P)$ *is a contraction mapping on* $(\mathcal{H}(X^d), h^d(m))$.

Theorem 6.9 *Let* $(X,m,w_1,\ldots,w_n,P)$ *be an RIFS. Then there exists a unique* n*-tuple* $A = (A_1, \ldots, A_n) \in \mathcal{H}(X)^n$ *such that* $w^n(A) = A$.

Definition 6.14 *Let* $(X,m,w_1,\ldots,w_n,P)$ *be an RIFS and* $A = (A_1, \ldots, A_n) \in \mathcal{H}(X)^n$ *the unique* n*-tuple such that* $w^n(A) = A$*. Then the attractor* $\mathcal{A}$ *of the RIFS is the set*

$$\mathcal{A} = \bigcup_{i=1}^{n} A_i. \tag{6.25}$$

Similar to the IFS case, we can state two decoding algorithms for a RIFS $(X,m, w_1,\ldots,w_n,P)$ (cf. [9]).

- The deterministic algorithm: Choose an arbitrary n-tuple $X_0 = (X_{0,1}, \ldots, X_{0,n}) \in \mathcal{H}(X)^n$. Obtain X_{k+1} for $k \in \mathbb{N}$ from X_k by computing $X_{k+1} = w^n(X_k)$. Let $\lim_{i\to\infty} X_i = (A_1, \ldots, A_n) \in \mathcal{H}(X)^n$. Then $\mathcal{A} = \bigcup_{i=1}^n A_i$.
- The chaos game: Choose an arbitrary element $x_0 \in X$ and an arbitrary index $i_0 \in \{1, \ldots, n\}$. Obtain x_{k+1} for $k \in \mathbb{N}$ from x_k by randomly choosing some index i_{k+1} such that $P(i_{k+1}, i_k) \neq 0$ and computing $x_{k+1} = w_{i_{k+1}}(x_k)$. Then the accumulation points of the sequence (x_k) are elements of $\mathcal{A}$ and the topological closure of the set of accumulation points equals $\mathcal{A}$ with probability 1.

We call a RIFS $(X,m,w_1,\ldots,w_n,P)$ affine, if the IFS $(X,m,w_1,\ldots,w_n)$ is affine. The attractors of affine RIFS can be computed using PWFA.

Theorem 6.10 *Let $(\mathcal{S}^d,m)$ for some $d \in \mathbb{N}^+$ be a complete metric space such that $\mathcal{S}$ is a semiring and let $\mathcal{U}$ be a non-empty compact subset of $\mathcal{S}^d$. Furthermore, let $(\mathcal{U},m,w_1,\ldots,w_n,P)$ be an affine RIFS with attractor $\mathcal{A}$. There exists a PWFA Z of dimension d over $\mathcal{S}$ such that $S(Z) = \mathcal{A}$.*

Proof: The matrix P is irreducible, thus for every column $j \in \{1,\ldots,n\}$ there is some row $i \in \{1,\ldots,n\}$ such that $P(i,j) \neq 0$. Let the smallest such i for each column j be denoted by $u(j)$. Further we define the successor function

$$s(i,j) = \begin{cases} i & \text{if } P(i,j) \neq 0 \\ u(j) & \text{otherwise} \end{cases} \tag{6.26}$$

for $(i,j) \in \{1,\ldots,n\}^2$. The automaton Z has $dn+n$ states and n labels, i.e. $Q_Z = \{1,\ldots,dn+n\}$ and $\Sigma_Z = \{1,\ldots,n\}$. Let v_0 denote an arbitrary element of $\mathcal{U}$. We choose F_Z as the vector $(v_0(1)\ \ldots v_0(d)\ 0\ \ldots\ 0\ 1\ 0\ldots\ 0)^T$, i.e. the vector that equals v_0 in the first d componenets, is one at component $dn+1$ and is zero otherwise. The initial distribution of Z for dimension k is 1 at the states $k,k+d,k+2d,\ldots,k+(n-1)d$ and zero otherwise. Let $w_k(x) = B_k x + c_k$ for $k = 1,\ldots,n$. Let $\mathcal{B}'(a,b,k) \in \mathcal{S}^{(dn+n)\times(dn+n)}$ be given by

$$\mathcal{B}'(a,b,k)(i,j) = \begin{cases} B_k(i-a+1,j-b+1) & \text{for } 1 \leq i-a+1, j-b+1 \leq d \\ 0 & \text{otherwise,} \end{cases} \tag{6.27}$$

for $1 \leq a,b \leq dn+n-d+1$ and $k = 1,\ldots,n$, i.e. the $(dn+n)\times(dn+n)$ matrix over $\mathcal{S}$ that has the block B_k at position (a,b) and is zero otherwise. Further let $\mathcal{B}(a,b,k) \in \mathcal{S}^{(dn+n)\times(dn+n)}$ be given as

$$\mathcal{B}(a,b,k) = \mathcal{B}'(d(a-1)+1,d(b-1)+1,k) \tag{6.28}$$

for $1 \leq a,b \leq n$ and $k = 1,\ldots,n$, i.e. instances of the matrices $\mathcal{B}'$ where the non-zero blocks are aligned to multiples of d shifted by one (as our indexes start at 1). Similarly let $\mathcal{C}'(a,b,k) \in \mathcal{S}^{(dn+d)\times(dn+d)}$ be given by

$$\mathcal{C}'(a,b,k)(i,j) = \begin{cases} c_k(i-a+1) & \text{for } j = (dn+b) \text{ and } 1 \leq i-a+1 \leq d \\ 0 & \text{otherwise} \end{cases} \tag{6.29}$$

for $1 \leq a \leq dn+n-d+1$, $1 \leq b \leq d$ and $k = 1,\ldots,n$, i.e. the $(dn+n)\times(dn+n)$ matrix over $\mathcal{S}$ that has the vector c_k in column $dn+b$ starting at row a and is zero otherwise. Furthermore, let $\mathcal{C}(a,b,k) \in \mathcal{S}^{(dn+d)\times(dn+d)}$ be given by

$$\mathcal{C}(a,b,k)(i,j) = \mathcal{C}'(d(a-1)+1,b,k) \tag{6.30}$$

for $1 \leq a \leq n$, $1 \leq b \leq d$ and $k = 1,\ldots,n$. Let $E(i,j) \in \mathcal{S}^{(dn+n)\times(dn+n)}$ denote the $(dn+n)\times(dn+n)$ matrix over $\mathcal{S}$ which is 1 at position (i,j) and zero otherwise for $1 \leq i,j \leq dn+n$. Then the transition matrix A_{Z_i} can be written as

$$A_{Z_i} = \sum_{j=1}^{n} (\mathcal{B}(s(i,j),j,s(i,j)) + \mathcal{C}(s(i,j),j,s(i,j)) + E(nd+s(i,j),nd+j)) \tag{6.31}$$

for $i = 1,\ldots,n$.

For any word $w \in \Sigma_Z^*$ the vector $\mu_Z(w)F$ has no more than $d+1$ non-zero components, which is similar to the IFS construction. There are n copies of the first d states used in the IFS construction and one additional state per copy, which corresponds to state $d+1$ of the IFS construction. The automaton starts in the first copy using an arbitrary vector contained in $\mathcal{U}$. Let $v = a_k \dots a_1 \in \Sigma_Z^*$ and assume that the vector $\mu_Z(v)F_Z$ has non-zero components lying in the q-th copy only, i.e. the system resides in the q-th copy. Furthermore assume that $v' = a_{k+1}a_k \dots a_1$ for some symbol $a_{k+1} \in \Sigma_Z$. If $P(a_{k+1}, q)$ is not zero, then the system will reside in the copy a_{k+1} after processing v'. Otherwise, if $P(a_{k+1}, q)$ is zero, then we are not allowed to apply the transformation $w_{a_{k+1}}$ in this situation. In this case we use the successor function to compute a valid map and target copy, i.e. we substitute a_{k+1} by pretending that the symbol we have encountered is $s(a_{k+1}, q)$. The argumentation that Z computes the attractor of the RIFS is analog to the IFS case. □

In our scenario, the real difference between IFS and RIFS is that the matrix P of an RIFS may have zero elements. This means that there are, in contrast to IFS, situations where we are not allowed to apply a certain transformation. As P is irreducible, there is at least one transformation we can apply at any time.

If we define the term image as a compact subset of some compact metric space (X, m), then the attractor $\mathcal{A}$ of an RIFS as given above is an image, i.e. the information whether each point of the space is inside or outside the attractor. This can be augmented by defining a normalized invariant measure (cf. [8, 17]) on X, which is an additive function f defined on the Borel subsets of X such that $f(X) = 1$.

We can implement something similar in the following way using a PWFA to obtain a real greyscale image from an RIFS $(\mathcal{K} \subset \mathbb{R}^2, m, w_1, \dots, w_n, P)$ where $\mathcal{K}$ is compact:

- Construct a PWFA Z representing the RIFS as described in the proof of Theorem 6.10 and add a dimension that computes the probability of each chosen path. The probability can be computed using a WFA. We have to adjust the probabilities, if we apply some kind of label substitution and encounter cases where the application of certain transformations is not allowed. More precisely that is: if we generate the transition from state s to state t of the Markov model for k different labels, then each of these transitions is assigned a probability of $\frac{p_t}{k}$.
- Choose a word length k.
- Compute the tuples $f(w) = (x_w, y_w, p_w)$ for each $w \in \Sigma_Z^k$.
- Let $Y = [\alpha, \beta] \times [\gamma, \delta]$ for real numbers $\alpha < \beta$ and $\gamma < \delta$ and M, N positive integers. We define the function $g : Y \mapsto \{1, \dots, M\} \times \{1, \dots, N\}$ as

$$g((x,y)) = \begin{cases} \left(\left\lfloor \frac{(M-1)(x-\alpha)}{\beta-\alpha} + \frac{1}{2} \right\rfloor + 1, \left\lfloor \frac{(N-1)(y-\gamma)}{\delta-\gamma} + \frac{1}{2} \right\rfloor + 1 \right) & \text{for } (x,y) \in Y \\ (0,0) & \text{otherwise} \end{cases} \tag{6.32}$$

  that assigns coordinates in $\{1, \dots, M\} \times \{1, \dots, N\}$ to the points of Y. Then the image $I_k = [0,1]^{M \times N}$ is given by

$$I_k(i,j) = \sum_{w \in \Sigma^k} p_w \delta((i,j), g(x_w, y_w)) \tag{6.33}$$

for $(i,j) \in \{1,\ldots,M\} \times \{1,\ldots,N\}$, where δ denotes the Kronecker δ.

Using the above algorithm, we can construct a sequence $(I_k)_{k=0}^{\infty}$ of images that converges to a limit image I. If the interval Y completely contains the attractor $\mathcal{A}$, then the sum of the elements of the image matrix I is 1. For this reason we will probably need to apply some scaling factor such that e.g. the maximal element of I is mapped to 1 in case of a floating point image or 255 in case of an 8 bit greyscale image. In practice we cannot compute the limit image I, but the evaluation for a sufficiently large k should deliver a suitable approximation.

6.3 Mutually recursive function systems

Mutually recursive function systems (MRFS, cf. [17, 18]) are an extension of the IFS concept. Let $(X,m,w_1,\ldots,w_n)$ denote an IFS. Throughout this section, we again assume that each transformation is affine. The successor set X_{k+1} of some compact set $X_k \subset X$ is defined as

$$X_{k+1} = w_1(X_k) \cup w_2(X_k) \ldots \cup w_n(X_k) \tag{6.34}$$

using the Hutchinson operator. This can be generalized by using q subsets $X_{k,i}$ of X for each k instead of one and defining a finite set of mutually recursive formulas as

$$X_{k+1,i} = w_{i_1}(X_{k,j_1}) \cup \ldots \cup w_{i_r}(X_{k,j_r}) \tag{6.35}$$

for $i = 1,\ldots,q$ and $(i_a,j_b) \in \{1,\ldots,n\} \times \{1,\ldots,q\}$ for $(a,b) \in \{1,\ldots,r\}^2$ where $r > 0$ depends on i. These q sets can then be divided into final or display variables, which are used for the construction of the attractor and non-display variables, which are only used for intermediate computations. We can express the structure of such an equation system as an NFA N with q states, i.e. there is one state for each variable. If the recursive definition of the variable $X_{k+1,i}$ contains the set $X_{k,j}$ transformed by w_a, then the automaton has an edge labeled by a starting at state j and going to state i. The states corresponding to final variables are marked as final states and every state is an initial state. The definition of MRFS allows that the states of N can be partitioned into unconnected subsets. If this is the case, then the attractor of the MRFS can be defined as the union of a set of attractors of MRFS, for which the automaton N is recurrent (i.e. each state can be reached from any other). We can thus without loss of generality assume that the NFA N is recurrent, as the set $\mathcal{D}(\mathcal{S},d)$ is closed under the set union operation for each semiring $\mathcal{S}$ and each dimension $d \in \mathbb{N}^+$. This assumption removes one structural difference between MRFS and RIFS. Another more important difference is that we do not require each single transformation w_j to be a contraction mapping. It is sufficient, if the resulting set of mutual recursive formulae satisfies the loop contraction property. The loop contraction property is given, if for each loop in the NFA N the corresponding composed transformation is contractive, i.e. if a loop is labeled by the word $v = a_1 \ldots a_k$ then $w_v = w_{a_1} \circ w_{a_2} \ldots \circ w_{a_k}$ is a contraction mapping. The above can be written shorter in the following definition (cf. [17]).

Definition 6.15 *A deterministic mutually recursive function system (DMRFS) is a compact metric space (X,m) together with a set of mutually recursive formulae*

of $q \geq 1$ variables, each variable defined as a union of images of some other variables under affine transformations on X, such that they satisfy a loop contractivity condition. Some variables are marked as final or display variables.

The q equations of a DMRFS define a kind of extended Hutchinson operator $W : \mathcal{H}(X)^q \mapsto \mathcal{H}(X)^q$. The operator W of each DMRFS has a unique fixed point in $\mathcal{H}(X)^q$, i.e. an assignment of variables $X_i, i = 1, \ldots, q$ such that

$$W((X_1, \ldots, X_q)) = (X_1, \ldots, X_q) \ . \tag{6.36}$$

If $(X_1, \ldots, X_q)$ denotes the fixed point of an MRFS, then the attractor $\mathcal{A}$ of the system is defined as

$$\mathcal{A} = \bigcup_{i \in F} X_i \ . \tag{6.37}$$

Similar to the IFS and RIFS case, the attractor is also given by

$$\mathcal{A} = \lim_{k \to \infty} \bigcup_{i \in F} X_{k,i} \tag{6.38}$$

for arbitrary non-empty compact subsets $X_{0,i}, i = 1, \ldots, q$ of X (cf. [17]).

DMRFS can be extended to probabilistic MRFS (PMRFS) by assigning probabilities to the application of transformations and thus the computation of a texture in addition to an image (cf. [17]).

The set of DMRFS computable sets has some interesting closure properties. It is easy to see that DMRFS are closed under affine transformations of the underlying space X. We introduce a new variable that becomes the only display variable and is defined as the union of the affine transformations of the previous display variables. A DMRFS M computing the set union of the sets computed by the DMRFS M_1 with q_1 variables and M_2 with q_2 variables over the same compact metric space can be constructed using $q_1 + q_2 + 1$ variables, where we keep the original recursive formulae and introduce one variable that becomes the single final variable and is defined as the union of the final variables in M_1 and M_2.

We introduce the projection operator $\mathcal{P}_k$ given in [17] and give a result proven in the same paper before we show that the attractors definable by DMRFS are PWFA computable.

Definition 6.16 *Let X be a set and k a positive natural number. Then the projection operator $\mathcal{P}_k(B) : \wp(X^k) \mapsto \wp(X)$ is defined as*

$$\mathcal{P}_k(B) = \{x | \exists (x_1, \ldots, x_k) \in B \text{ such that } x = x_r \text{ for some } 1 \leq r \leq k\} \tag{6.39}$$

for each $B \in \wp(X^k)$.

Theorem 6.11 *Let M be a DMRFS (or a PMRFS) on q variables and let X the underlying compact metric space. Then there is a higher-dimensional IFS I whose projected attractor $\mathcal{P}_q(\mathcal{A}_I)$ is the same as that of M.*

Theorem 6.12 *Let $(\mathcal{S}^d, m)$ for some $d \in \mathbb{N}^+$ denote a complete metric space such that $\mathcal{S}$ is a semiring and let $\mathcal{U}$ be a compact subset of $\mathcal{S}^d$. Furthermore, let M denote a DMRFS with q variables over $(\mathcal{U}, m)$ with attractor $\mathcal{A}_M$. There is a PWFA Z of dimension d over $\mathcal{S}$ such that $S(Z) = \mathcal{A}_M$.*

Proof: According to Theorem 6.11 the attractor $\mathcal{A}_M$ can be represented as the projection $\mathcal{P}_q$ of an IFS over $\mathcal{S}^{dq}$. The projection operator $\mathcal{P}_q : X^q \mapsto X$ can be rewritten as

$$\begin{array}{rcl} \mathcal{P}_q(B) & = & \{x|\exists(x_1,\ldots,x_q) \in B \text{ such that } x = x_r \text{ for some } 1 \leq r \leq q\} \\ & = & \bigcup_{i=1}^{q} \{x|\exists(x_1,\ldots,x_q) \in B \text{ such that } x = x_i\} \end{array} \tag{6.40}$$

for each $B \subset X^q$. The vector spaces $\mathcal{S}^{dq}$ and $\mathcal{S}^{dq}$ are trivially isomorphic. Let Z' be a PWFA that represents the IFS over $\mathcal{S}^{dq}$ using the construction given above, which produces ST-consistent automata. Let $U_{dq,q,k} \in \mathcal{S}^{d\times dq}$ be given by

$$U_{dq,q,k}(i,j) \begin{cases} 1 & \text{if } (i,j) = (p, d(k-1)+p) \text{ for } 1 \leq p \leq d \\ 0 & \text{otherwise,} \end{cases} \tag{6.41}$$

i.e. a matrix that projects a vector $v \in \mathcal{S}^{dq}$ to its components $d(k-1)+1,\ldots,dk$. Then the attractor $\mathcal{A}_M$ can be computed as

$$\mathcal{A}_M = \bigcup_{k=1}^{q} U_{dq,q,k} S(Z') = \bigcup_{k=1}^{q} S(\Xi[U_{dq,q,k} I_{Z'}](Z')) = S(Z) \quad . \tag{6.42}$$

Note that unlike the situation we will discuss in Remark 7.2, the application of the projection matrices is in this case no problem, as Z' is ST-consistent. □

As in the case of RIFS we can introduce an additional component that assigns a probability to each computed point and thus compute a textured image instead of a pure image.

The set computed by a DMRFS is always compact and there exist PWFA computable sets, which are not compact (e.g. the exponential function). Thus the set of DMRFS computable sets is a strict subset of the set of PWFA computable sets.

Chapter 7

Relations and functions

When we use WFA to compute real functions, the input word is interpreted as the argument of the function. The choice of the alphabet $\{0,1\}$ and the interpretation of input words using the functions $q[2]$ and $r[2]$ is very common. Thus we will use the term *a WFA computing a real function* as an abbreviation for the term *a WFA computing a real function under* $r[2]$ in the following. As PWFA are WFA (at least if we consider the word function), we similarly can use PWFA to compute real functions while interpreting the input word as the argument of the function (transforming a WFA computing a real function into a PWFA computing the same function is however not always straight-forward, see Remark 7.1), but we can also compute relations and functions by explicitly computing the set of vectors that are contained in the relation or function respectively (remember that a relation is a subset of a Cartesian product and a function is a binary relation satisfying some conditions, cf. [76]).

Definition 7.1 *Let $\mathcal{S}$ be a semiring and let $\mathcal{R} \subset \mathcal{S}^p \times \mathcal{S}^q$ denote a relation for some positive natural numbers p and q. We say that a PWFA Z of dimension $p+q$ over $\mathcal{S}$ computes $\mathcal{R}$, if*

$$S(Z) = \{(x_1,\ldots,x_p,y_1,\ldots,y_q) | ((x_1,\ldots,x_p),(y_1,\ldots,y_q)) \in R\} \quad . \tag{7.1}$$

Likewise we say that a PWFA Z of dimension $p+q$ over $\mathcal{S}$ computes the function $\mathcal{F} : \mathcal{S}^p \mapsto \mathcal{S}^q$ over the semiring $\mathcal{S}$ for positive natural numbers p and q, if Z computes the relation underlying $\mathcal{F}$.

Remark 7.1 *If we are given a real WFA X that computes a certain real function f, then it is only straight-forward to construct a real PWFA Y of dimension 2 computing f by transforming X, if f is defined everywhere in the unit interval $[0,1]$. If e.g. the values produced by f_X are bounded by some real number r but f_X is undefined for some points in the unit interval, then the PWFA Y obtained from X by choosing the first component as a WFA computing the function $q[2]$ and the second component as X will produce erroneous values, because due to the Bolzano-Weierstraß theorem each bounded sequence has an accumulation point. As we are not considering any non-total WFA computable real functions in this thesis, we will not tackle this problem. If we would change the definition of the set computed by a PWFA to a real WFA style convergence, then this problem would vanish, but we likely would loose a large part of the expressiveness of the model.*

When we use PWFA to compute functions, we are mostly interested in automata that show a certain well-defined behavior, which we express in the following definition.

Definition 7.2 *Let $(\mathcal{S}^d, m)$ for some $d \in \mathbb{N}^+$ be a complete metric space such that $\mathcal{S}$ is a semiring. Furthermore, let X be a PWFA of dimension d over the $\mathcal{S}$. We say that X properly produces the set $S \subset \mathcal{S}^d$, if X is ST-consistent and $S(X) = S$. Likewise we say that X properly computes the relation $\mathcal{R}$ or function $\mathcal{F}$, if it properly computes the set underlying $\mathcal{R}$ or $\mathcal{F}$ respectively.*

7.1 Multidimensional separable functions and delay functions

Multidimensional separable functions $p : [0,1]^d \mapsto [0,1]$ of dimension $d > 1$ can be implemented by WFA in two fashions. The first is to increase the number of input symbols from 2 to 2^d. This approach is used in [20]. The second way is to keep an alphabet cardinality of 2 and multiplex the digits of d binary input words into a single input word. It can be found in [50] for $d = 2$, where binary trees were used instead of quadtrees to partition images for WFA based image compression. A general approach is given in the following theorem.

Definition 7.3 *Let Σ be an alphabet and $k, m \in \mathbb{N}, m > 0, 1 \leq k \leq m$.*

1. *Let $w = a_1 \dots a_n \in \Sigma^*$. We call the word*

$$o(k,m) = \begin{cases} a_k a_{k+m} \dots a_{k+m\lfloor \frac{n-k}{m} \rfloor} & \text{if } n \geq k \\ \varepsilon & \text{otherwise} \end{cases} \tag{7.2}$$

   the k projection of w modulo m.

2. *Let $w = a_1 a_2 \dots \in \Sigma^\omega$. We call the word*

$$o(k,m) = a_k a_{k+m} a_{k+2m} \dots \tag{7.3}$$

   the k projection of w modulo m.

Definition 7.4 *Let Σ be an alphabet, S a set, $k, m \in \mathbb{N}, m > 0, 1 \leq k \leq m$ and $f : \Sigma^* \mapsto S$ or $f : \Sigma^\omega \mapsto S$ a function. We call the function*

$$f(k,m)(w) = f(o(k,m)(w)) \tag{7.4}$$

the (k,m) delay function of f.

Lemma 7.1 *Let X be a WFA over the semiring $\mathcal{S}$. Let $k, m \in \mathbb{N}, m > 0, 1 \leq k \leq m$. There is a WFA Y over $\mathcal{S}$ computing the (k,m) delay function of f_X.*

Proof: Assume that $Q_X = \{1, \dots, n_X\}$ for some $n_X \in \mathbb{N}^+$. Y is constructed by producing m copies of X that are visited in turn when reading the input word. Y is formally defined as

- $Q_Y = \{1, \dots, mn_X\}$,

- $\Sigma_Y = \Sigma_X$,
- $A_{Y_i} = \begin{pmatrix} O & B_{i,1} & O & \dots & O \\ O & O & B_{i,2} & \dots & O \\ & & \vdots & & \\ O & O & O & \dots & B_{i,m-1} \\ B_{i,m} & O & O & \dots & O \end{pmatrix}$
  for $i \in \Sigma_Y$, where O denotes the zero matrix in $\mathcal{S}^{n_X \times n_X}$ and

$$B_{i,j} = \begin{cases} A_{X_i} & \text{if } j = k \\ U_{n_X} & \text{otherwise} \end{cases} \tag{7.5}$$

  where U_{n_X} denotes the identity matrix of $\mathcal{S}^{n_X \times n_X}$,
- columns 1 to n_X of I_Y are I_X, the rest of the elements of I_Y is zero and
- $F_Y = (\ F_X^T \quad F_X^T \quad \dots \quad F_X^T\ )^T$.

□

Lemma 7.2 *Let X be a PWFA of dimension d over some semiring $\mathcal{S}$. Let $k,m \in \mathbb{N}, m > 0, 1 \leq k \leq m$. Then every PWFA Y of dimension d over $\mathcal{S}$ computing the (k,m) delay function of X satisfies $S(Y) = S(X)$ and $T(Y) = T(X)$.*

Proof: Let $v \in \mathcal{S}^d$. If v is produced by f_X j times (where j may be infinite), then f_Y produces v at least j and at most mj times. As $\bigcup_{w\in\Sigma^*}\{f_Y(w)\} = \bigcup_{w\in\Sigma^*}\{f_X(w)\}$, we obtain $T(Y) = T(X)$ and $S(Y) = S(X)$. □

Definition 7.5 *Let $Z = \{Z_1, \dots Z_m\}$ denote a finite set of PWFA of dimension $p+q$ over $\mathbb{C}$ computing the relations $\mathcal{R}_j \subset \mathbb{C}^p \times \mathbb{C}^q$ for positive natural numbers p,q and $j = 1, \dots, m$. We say that Z is consistent if*

- *Z_j computes $S(Z_j)$ properly (i.e. Z_j is ST-consistent) for $j = 1, \dots, m$ and*
- *$\Sigma_{Z_1} = \Sigma_{Z_2} = \dots = \Sigma_{Z_m}$.*

Let Z_i' be the automaton obtained from Z_i by projection on the dimensions 1 to p for $i = 1, \dots, m$. If $\mathcal{R}_i$ is a function, then as usual we call $S(Z_i')$ the domain of $\mathcal{R}_i$ for $i = 1, \dots, m$.

Theorem 7.1 *Let $\mathcal{F}_1, \dots, \mathcal{F}_m : \mathbb{C}^p \mapsto \mathbb{C}^q$ be PWFA computable functions computed by the PWFA $Z_1, \dots, Z_m$ for positive natural numbers p,q, where the system $Z = \{Z_1, \dots, Z_m\}$ is consistent. Let I_i denote the domain of the function $\mathcal{F}_i$ for $i = 1, \dots, m$. There is a PWFA P of dimension $mp+q$ over $\mathbb{C}$ that computes the relation*

$$S(P) = \left\{ \begin{array}{ll} (t_{1,1}, t_{1,2}, \dots, t_{m,p}, & \\ & \prod_{k=1}^m \mathcal{F}_k((t_{k,1}, t_{k,2}, \dots, t_{k,p}))(1) \\ & \prod_{k=1}^m \mathcal{F}_k((t_{k,1}, t_{k,2}, \dots, t_{k,p}))(2) \\ & \dots \\ & \prod_{k=1}^m \mathcal{F}_k((t_{k,1}, t_{k,2}, \dots, t_{k,p}))(q)) \\ & |(t_{i,1}, \dots, t_{i,p}) \in I_i \text{ for } i = 1, \dots, m \end{array} \right\}. \tag{7.6}$$

Proof: Let P'_k denote a PWFA computing the (k,m) delay function of f_{Z_k} for $k = 1, \ldots, m$. We choose the components $(k-1)p+1$ to kp of P as the components 1 to p of P'_k for $k = 1, \ldots, m$. Thus the convergence behavior of the first mp components of P is clear, as it is directly implied by the behavior of the Z_k, which means that the projection of P to its first mp components is an ST-consistent automaton. Let P'_{kj} denote the PWFA obtained from P'_k by reducing it to component j. Then we choose the components $mp+j$ of P as the product of the PWFA P'_{kp+j} for $k = 1, \ldots, m$ and $j = 1, \ldots, q$ (in the sense of the Hadamard products of their word functions). Clearly, if we construct P in this fashion, then $S(P)$ contains the set stated in equation 7.6. It remains to show that $S(P)$ does not contain any additional vectors. Let $w_1, w_2, \ldots$ be a sequence of words of increasing length in Σ^*. Assume that $\lim_{i\to\infty} f_{P_{k_1 j}'}(w_i)$ does not exist but $\lim_{i\to\infty} f_{P_{k_1 j}'}(w_i) f_{P_{k_2 j}'}(w_i)$ does exist for some $(k_1,k_2) \in \{1, \ldots, m\}^2$ and some $j \in \{p+1, \ldots, p+q\}$. As the Z_k are all ST-consistent, the values produced by $f_{P_{k_1 j}'}$ along the sequence w_i become arbitrarily close to values that are elements of $S(P_{k_1 j}')$. This implies that the value $\lim_{i\to\infty} f_{P_{k_1 j}'}(w_i) f_{P_{k_2 j}'}(w_i)$ is already produced for a sequence of words for which $P'_{k_1 j}$ converges. □

As WFA are closed under addition, we obtain the following corollary from Theorem 7.1.

Corollary 7.1 *Let $n \in \mathbb{N}$ and $a_{i_1,i_2,\ldots,i_m}$ real numbers for $0 \le i_j \le n, j = 1, \ldots, m$. Let $\mathcal{F}_1, \ldots, \mathcal{F}_m$ total functions computable by real WFA on $[0,1]$. There is a PWFA X that computes the set*

$$S(X) = \left\{ \left(t_1, \ldots, t_m, \sum_{i_1=0}^{n} \cdots \sum_{i_m=0}^{n} a_{i_1,\ldots,i_m} \prod_{j=1}^{m} \mathcal{F}(t_j)^{i_j} \right) \middle| (t_1, \ldots, t_m) \in [0,1]^m \right\} . \tag{7.7}$$

This enables us to compute splines, which we will discuss in Chapter 8.

7.2 Real relations and functions

7.2.1 Real relations over closed finite real intervals

As we have seen in section 3.2, WFA over the real numbers can compute real polynomials on the unit interval. This means that PWFA of dimension d are able to compute each relation

$$S = \{((p_1(t), p_2(t), \ldots, p_m(t)) | t \in [0,1]\} \tag{7.8}$$

where the p_i are real polynomials for $i = 1, \ldots, m$. An example is shown in Figure 4.1, where the computed set is

$$S = \{(t^3 - t^2, t^2 - t) | t \in [0,1]\} . \tag{7.9}$$

If $p(x) : \mathbb{R} \mapsto \mathbb{R}$ is a real polynomial, then $p(sx+t)$ is also a real polynomial for $s,t \in \mathbb{R}$. This means that we can compute the evaluation of each polynomial on each real interval $[a,b]$ using a PWFA.

Corollary 7.2 *Let p be a real polynomial. Then there is a PWFA Z over $\mathbb{R}$ that computes the set*

$$S(Z) = \{(c,d)|c \in [a,b], d = p(c)\} \tag{7.10}$$

for the real interval $[a,b]$.

Proof: If $b < a$, then $S(Z) = \emptyset$, which is clearly PWFA computable. Now assume that $b \geq a$. If we transform $S(Z)$ to

$$S(Z) = \{a + (b-a)t, p(a + (b-a)t))|t \in [0,1]\} \tag{7.11}$$

then both components are total real WFA computable functions. □

Another family of functions that can be computed by PWFA on closed finite real intervals is the set of inverse functions of polynomials, which we obtain by swapping the components of a suitable PWFA.

We show in the next paragraph that PWFA do not gain any computational power in terms of computing smooth (that means every derivative is continuous) real functions, if we allow them to read the input words in reversed order compared to the usual interpretation of input words for WFA computing real functions. This result was first published in [90]. As we do a comparison of a family of PWFA to WFA computing real functions, we will be using alphabets starting at 0 instead of the usual alphabets starting at 1 for the rest of the subsection. Let $\Sigma = \{0, \ldots, k-1\}$ where $k \in \mathbb{N}, k \geq 2$ and let w^R denote the reversed word of w for each $w \in \Sigma^*$. Then we obviously have

$$\overline{\bigcup_{w \in \Sigma^*} q[k](w^R)} = \overline{\bigcup_{w \in \Sigma^*} q[k](w)} = [0,1] \ , \tag{7.12}$$

so we are considering PWFA computing real functions on the real unit interval $[0,1]$.

Let $d \in (0,1)$. Then we define the iterated function system (IFS, see section 6.1) $D(d)$ as the set of functions

$$\begin{array}{rcl} d_0(x) & = & dx \\ d_1(x) & = & dx + (1-d) \ . \end{array} \tag{7.13}$$

The system $D(d)$ has the attractor $[0,1]$. We first show that polynomials on the unit interval can be computed while using this IFS for the first component of a PWFA.

Lemma 7.3 *Let $k \in \mathbb{N}$ and $d \in (0,1)$. There is a PWFA X of dimension 2 over $\mathbb{R}$ that computes the set $\{(x,x^k)|x \in [0,1]\}$, while the first component of X is computed as the IFS $D(d)$.*

Proof: The automaton for the IFS $D(d)$ is shown in Figure 7.1. The recursion for x^k can be written as

$$\begin{array}{rcl} e_0(x) & = & (dx)^k = d^k x^k \\ e_1(x) & = & (dx + (1-d))^k = \sum_{i=0}^{k} \binom{k}{i} d^i x^i (1-d)^{k-i} \ . \end{array} \tag{7.14}$$

The construction scheme is shown in Figure 7.2. Thus the automaton X can be built by combining the automata shown in Figure 7.1 and Figure 7.2. □

As WFA are closed under sum and multiplication by a constant, we obtain the following.

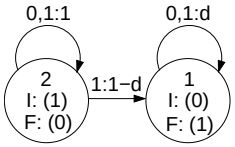


Figure 7.1: IFS automaton for system $D(d)$.

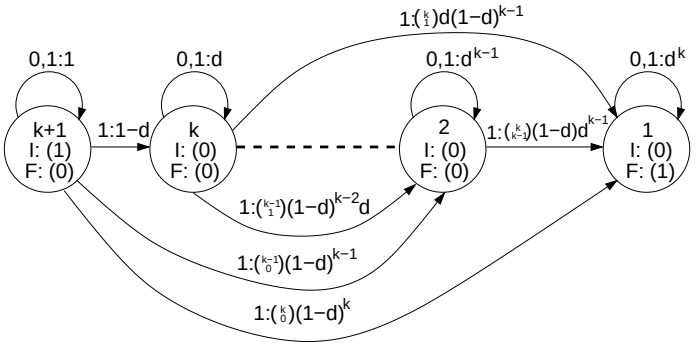


Figure 7.2: Automaton computing second component x^k for first component displayed by IFS $D(d)$.

Theorem 7.2 *Let $p(x) = \sum_{i=0}^{k} a_i x^i$ be a real polynomial for $k \in \mathbb{N}$. For each $d \in (0,1)$ there is a PWFA X that computes the set $\{(x, p(x)) | x \in [0,1]\}$ while computing the first component as the IFS $D(d)$.*

It is remarkable that unlike the usual WFA construction for polynomials we cannot use one single line-automaton (cf. [20]) to represent a complete polynomial with more than one non-zero coefficient. Clearly the construction for $x^{k+1}, k \in \mathbb{N}$ contains the construction for x^k as a sub-graph. It differs in two features:

1. The sub-automaton for x^k contained in x^{k+1} has outgoing edges.
2. When we build the automaton for x^{k+1} based on the automaton for x^k, the final distribution is changed instead of the initial distribution.

We now show that polynomials are the only smooth (that means having all derivates everywhere on the unit interval) real functions computable by PWFA using the IFS $D(1/2)$ to compute the first component. This is done by reducing such automata to real WFA. It is well known that the only smooth real functions computable by real WFA are polynomials (cf. [36]). If we interpret ω words over $\{0,1\}$ using the function $r[2]$, then the words with prefix 0 are interpreted as numbers in $[0,1/2]$ and the words with prefix 1 as numbers in $[1/2,1]$. This means that the unit interval is split into two halves by the first symbol. The choice of $1/2$ as the splitting point corresponds to the positional notation of numbers. We could however choose a different splitting point in $(0,1)$. This would result in a different (ω) word interpretation function under which we could still define WFA computing polynomials on the real unit interval. We conjecture that polynomials are still the only smooth real functions computable

by real WFA under this modified (ω) word interpretation function, however the proof given in [36] only considers the canonical case of a splitting point of $1/2$.

Lemma 7.4 *Let X be a real PWFA computing the smooth real function f on the unit interval as the set $S(X) = \{(x, f(x)) | x \in [0,1]\}$ while computing the first component as the IFS $D(1/2)$. There is a real WFA Y that computes the same function.*

Proof: Let $w = a_1 a_2 \ldots a_m \in \Sigma_X^*$. The first component of the result vectors of X is computed as

$$\begin{aligned} x(w) &= \tfrac{1}{2}\left(\left(\ldots \tfrac{1}{2}\left(\tfrac{1}{2}a_1\right) + \tfrac{1}{2}a_2 \ldots\right) + \tfrac{1}{2}a_{m-1}\right) + \tfrac{1}{2}a_m \\ &= \textstyle\sum_{i=1}^{m} 2^{-(m-i+1)} a_i \ . \end{aligned} \tag{7.15}$$

That means the input order is reversed in comparison to WFA, as apparently $x(w^R) = q[2](w)$ and $q[2](w^R) = x(w)$ hold for each w in Σ_X^*. Without loss of generality we assume that the set of states X uses to compute its first component is disjoint from the set of states it uses to compute the second. Then we can decompose X into two independent WFA computing the components of the result vectors of X. Let these sub-graphs of X be named Y' for the first component and Y'' for the second. Then equation 7.16 holds for $w = a_1 a_2 \ldots a_m \in \Sigma_X^*$.

$$\begin{aligned} (x(w), f(x(w))) &= (I_{Y'} \textstyle\prod_{i=1}^{m} A_{Y'_{a_i}} F_{Y'}, I_{Y''} \prod_{i=1}^{m} A_{Y''_{a_i}} F_{Y''}) \\ &= \left(I_{Y'} \left(\textstyle\prod_{i=1}^{m} A^T_{Y'_{a_{m-i+1}}} \right)^T F_{Y'}, I_{Y''} \left(\prod_{i=1}^{m} A^T_{Y''_{a_{m-i+1}}} \right)^T F_{Y''} \right) \\ &= \left(F^T_{Y'} \left(\textstyle\prod_{i=1}^{m} A^T_{Y'_{a_{m-i+1}}} \right) I^T_{Y'}, F^T_{Y''} \left(\prod_{i=1}^{m} A^T_{Y''_{a_{m-i+1}}} \right) I^T_{Y''} \right) \\ &= \left(q[2](w^R), F^T_{Y''} \left(\textstyle\prod_{i=1}^{m} A^T_{Y''_{a_{m-i+1}}} \right) I^T_{Y''} \right) \end{aligned} \tag{7.16}$$

Let Y''_R denote the sub-graph that is obtained from Y'' by transposing the transition matrices and swapping the initial and final distribution. It remains to show that Y''_R can be transformed into a WFA Y that computes a function that is defined everywhere. There are two problems we can encounter:

1. The function $r[2]$ maps the words $w_0 = w01^\omega$ and $w_1 = w10^\omega$ over $\{0,1\}^\omega$ to the same real number. Let $(v_{0_i})_{i=1}^{\infty} = w, w0, w01, w011, \ldots$ and $(v_{1_i})_{i=1}^{\infty} = w, w1, w10, w100, \ldots$. X produces the point $(r[2](w_0), f(r[2](w_0)))$, if there exists a subsequence (u_{0_i}) of (v_{0_i}) or (u_{1_i}) of (v_{1_i}) such that one of the two limits $\lim_{i\to\infty} f_{Y''_R}(u_{0_i})$ and $\lim_{i\to\infty} f_{Y''_R}(u_{1_i})$ exists. f_Y however has to be defined for w_0 as well as w_1.

2. Let $w = a_1 a_2 \ldots \in \{0,1\}^\omega$. Furthermore, let $(v_i)_{i=0}^{\infty} = \varepsilon, a_1, a_1 a_2, \ldots$ be the sequence of prefixes of w. It suffices that $\lim_{i\to\infty} f_{Y''_R}(u_i)$ exists for a subsequence (u_i) of (v_i) so that X produces the point $(r[2](w), f(r[2](w)))$. f_Y however has to converge for the complete sequence, i.e. $\lim_{i\to\infty} f_Y(v_i)$ has to exist.

We will show that both of these problems can be remedied by changing Y''_R.
Consider the first problem. Let $w \in \Sigma_X^*$, $w_0 = w01^\omega$ and $w_1 = w10^\omega$. Furthermore, let $(v_{0_i})_{i=1}^{\infty} = w, w0, w01, w011, \ldots$ and $(v_{1_i})_{i=1}^{\infty} = w, w1, w10, w100, \ldots$ be the sequences of prefixes of w_0 and w_1 respectively. If there are subsequences $(u_{0_i})_{i=1}^{\infty}$

of (v_{0_i}) and $(u_{1_i})_{i=1}^{\infty}$ of (v_{1_i}) such that $\lim_{i\to\infty} f_{Y_R''}(u_{0_i})$ and $\lim_{i\to\infty} f_{Y_R''}(u_{1_i})$ both exist, then they are equal, as X computes the function f. If for one of the two cases there is no converging subsequence, then the Bolzano-Weierstraß theorem implies that the absolute value of $f_{Y_R''}$ approaches infinity for the corresponding sequence, because if it were bounded, it would possess an accumulation point. This implies that there exist pairs (s,S) of symbols $s \in \{0,1\}$ and maximal cyclic sub-spaces S of $\mathbb{R}^{n_{Y_R''}}$ such that

- $(vA_{Y_{R\,s}''})^T \in S$ for each $v^T \in S$ and
- for each $v^T \in S$ exists some $k \in \mathbb{N}$ such that $|vA_{Y_{R\,s}''}^k F| > |2vF|$.

We consider a single pair (s,S) of the pairs mentioned above. Let $\{b_1,\ldots,b_k\}$ denote a basis of S and $\mathcal{B} = \{b_1,\ldots,b_k,r_{k+1},\ldots,r_{n_{Y_R''}}\}$ a basis of $\mathbb{R}^{n_{Y_R''}}$. We transform Y_R'' so that the sub-space S of dimension k corresponds to the states $1,\ldots,k$. Let B denote the matrix that transforms the standard basis to the basis $\mathcal{B}$. Then we substitute $I_{Y_R''}$ by $I_{Y_R''}B^{-1}$, $F_{Y_R''}$ by $BF_{Y_R''}$ and $A_{Y_{R\,i}''}$ by $BA_{Y_{R\,i}''}B^{-1}$ for $i = 0,1$. In this transformed automaton it is easy to see that we can use a simple NFA N to decide, whether the transformed automaton has assumed a configuration that belongs to the sub-space S of the original automaton. Now it is simple to substitute the non-convergent behavior of Y_R'' by the corresponding convergent behavior. If the NFA N enters any state corresponding to S, then Y_R'' starts to emulate the dual case while continuing the original computation masked by a final distribution of zero. If N no longer assumes any state corresponding to S, then Y_R'' resumes the original computation. If we repeat this for every present pair (s,S), then we will have removed the first of the two problems. As the number of states in Y_R'' is finite, there can only be a finite number of such pairs.
Now consider the second problem. Let $w = a_1a_2\ldots \in \Sigma_X^{\omega}$. Assume that there exists a subsequence $(u_i)_{i=1}^{\infty}$ of the sequence $(v_i)_{i=0}^{\infty} = \varepsilon, a_1, a_1a_2, \ldots$ for which $f_{Y_R''}$ converges. Furthermore, assume however that $\lim_{i\to\infty} f_{Y_R''}(v_i)$ does not exist. As above, the Bolzano-Weierstraß theorem guarantees that there is exactly one neighborhood that contains an infinite number of points produced by $f_{Y_R''}$ for (v_i). Each sequence of computed values that has an infinite number of values that are not contained in this neighborhood is unbounded, i.e. its absolute value approaches infinity. The automaton has to store the numbers responsible for this approaching to infinity in certain states at each stage of the computation, i.e. the appearance of such a number in a final state can be expressed as a regular language L. We can assume that $\varepsilon \notin L$. Then we can modify Y_R'' in a way so that if it reads the word $v_i \in L$, then $f_{Y_R''}$ computes the value $f_{Y_R''}(v_j)$ instead of $f_{Y_R''}(v_i)$, where j is the greatest integer smaller than i such that $v_j \notin L$. This means that we can also remove the second problem from Y_R'', which completes the proof. □

Every smooth real function computable by a WFA is a polynomial, so Theorem 7.3 follows.

Theorem 7.3 *Let X be a PWFA computing the smooth real function f on the unit interval as the set $S(X) = \{(x, f(x)) | x \in [0,1]\}$ while computing the first component as the IFS $D(1/2)$. Then f is a polynomial.*

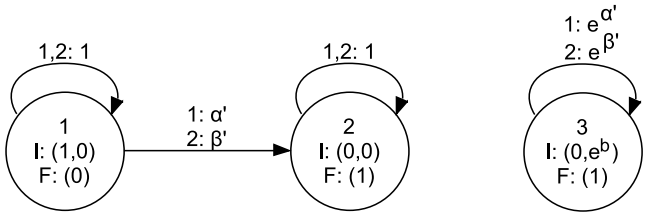


Figure 7.3: Automaton computing function e^{ax+b}, where $\alpha' = a\alpha$ and $\beta' = a\beta$ for $\alpha\beta < 0, \frac{\alpha}{\beta} \notin \mathbb{Q}$.

7.2.2 Trigonometric functions on $\mathbb{R}$

The computation of the exponential, sine and cosine function on $\mathbb{R}$ using PWFA was shown in [4]. We provide a few simple generalizations in this subsection.

Theorem 7.4 *Let $a,b \in \mathbb{R}$. Then there is a PWFA Z that computes the function $f(x) = e^{ax+b}$ by computing the set*

$$S(Z) = \left\{ \left(x, e^{ax+b}\right) \middle| x \in \mathbb{R} \right\} \quad . \tag{7.17}$$

Proof: The construction given in [4] computes the function e^x as the set

$$\overline{\{(n\alpha + m\beta, e^{n\alpha}e^{m\beta}) | n,m \in \mathbb{N}\}}$$

for $\alpha\beta < 0, \frac{\alpha}{\beta} \notin \mathbb{Q}$. We can extend this by writing

$$\begin{aligned} S(Z) &= \overline{\{(n\alpha + m\beta, e^{a(n\alpha+m\beta)+b}) | n,m \in \mathbb{N}\}} \\ &= \overline{\{(n\alpha + m\beta, e^{b}e^{n(a\alpha)+m(a\beta)}) | n,m \in \mathbb{N}\}} \\ &= \overline{\{(n\alpha + m\beta, e^{b}e^{n(\alpha')+m(\beta')}) | n,m \in \mathbb{N}\}} \end{aligned} \tag{7.18}$$

where $a' = a\alpha$ and $\beta' = a\beta$. The modified automaton is shown in Figure 7.3. □

Theorem 7.5 *Let $a,b \in \mathbb{R}$. Then there exists a PWFA X that computes the function $f(x) = sin(ax+b)$ by computing the set*

$$S(X) = \{(x, \sin(ax+b)) | x \in \mathbb{R}\} \tag{7.19}$$

and a PWFA Y that computes the function $g(x) = cos(ax+b)$ by computing the set

$$S(Y) = \{(x, \cos(ax+b)) | x \in \mathbb{R}\} \quad . \tag{7.20}$$

Proof: We proof the case of $f(x) = \sin(ax+b)$ for real numbers a and b, the case of $g(x) = \cos(ax+b)$ is analogous. The sine function is computed as the set

$$\overline{\{(n\alpha + m\beta, \sin(n\alpha + m\beta) | n,m \in \mathbb{N}\}}$$

for real numbers α and β such that $\alpha\beta < 0$ and $\frac{\alpha}{\beta} \notin \mathbb{Q}$ in [4]. We can thus write

$$\begin{aligned} S(X) &= \overline{\{(n\alpha + m\beta, \sin(a(n\alpha + m\beta) + b)) | n,m \in \mathbb{N}\}} \\ &= \overline{\{(n\alpha + m\beta, \sin((na\alpha + ma\beta) + b)) | n,m \in \mathbb{N}\}} \\ &= \overline{\{(n\alpha + m\beta, \sin((n\alpha' + m\beta') + b)) | n,m \in \mathbb{N}\}} \\ &= \overline{\{(n\alpha + m\beta, \sin(n\alpha' + m\beta')\cos(b) + \cos(n\alpha' + m\beta'))\sin(b)) | n,m \in \mathbb{N}\}} \end{aligned} \tag{7.21}$$

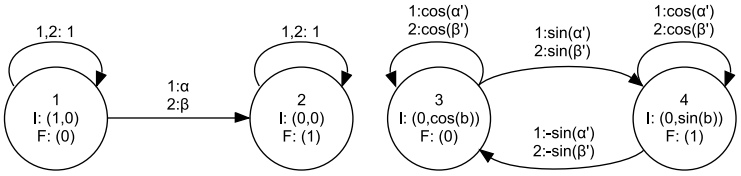


Figure 7.4: Automaton computing the function $sin(ax+b)$ for real numbers a and b where $\alpha\beta < 0$, $\frac{\alpha}{\beta} \notin \mathbb{Q}$, $\alpha' = a\alpha$ and $\beta' = a\beta$.

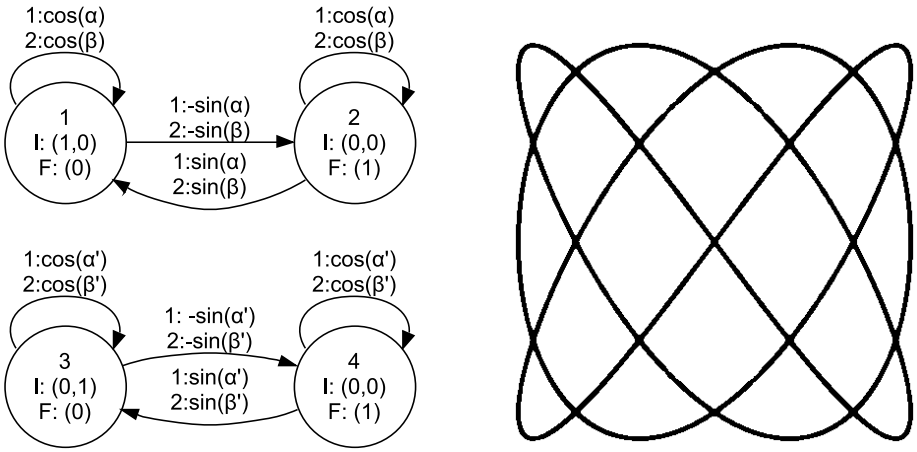


Figure 7.5: We choose real numbers α and β such that $\alpha\beta < 0, \frac{\alpha}{\beta} \notin \mathbb{Q}$ and derive $\alpha' = \cos\left(\frac{3}{4}\text{acos}(\alpha)\right)$ and $\beta' = \cos\left(\frac{3}{4}\text{acos}(\beta)\right)$. Then the automaton on the left computes the Lissajous figure given by the relation $\mathcal{L}(1,1,3,4,0)$. The image it produces is shown on the right.

for $\alpha' = a\alpha$ and $\beta' = a\beta$. The modified automaton is shown in Figure 7.4. □

Example 7.1 *Apart from the circle that can be computed by PWFA, Theorem 7.5 allows us to construct real PWFA computing Lissajous figures. Such figures are defined by the relations*

$$\mathcal{L}(A,B,a,b,\delta) = \{(A\sin(at+\delta), B\sin(bt))|t \in \mathbb{R}\} \tag{7.22}$$

for the real parameters A,B,a,b and δ. An example is shown in Figure 7.5.

If the set $\mathcal{D}(\mathbb{R})$ is closed under the set intersection operation, which we conjecture it is not, then we can also compute a set describing the tangent using a PWFA.

Theorem 7.6 *Assume that $\mathcal{D}(\mathbb{R})$ is closed under intersection. Then there is a PWFA Z that computes the set*

$$S(Z) = \left\{(x,tan(x),0)|x \in \mathbb{R} \setminus \left\{\frac{\pi}{2} + k\pi | k \in \mathbb{Z}\right\}\right\} \quad . \tag{7.23}$$

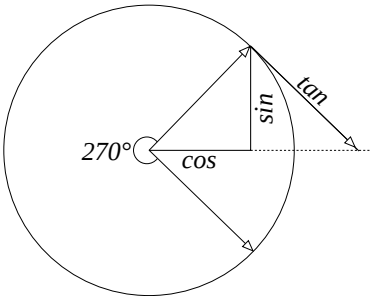


Figure 7.6: The direction of the tangent is obtained for a point by rotating it by 270° around the origin of the Cartesian plane.

Proof: Consider the construction of the unit circle given in [4]. A vector describing the direction of the tangent for each point can be obtained by applying a 270° rotation. This is depicted in Figure 7.6. The tangent of some angle is given by the distance from the point corresponding to the angle on the unit circle to the point where the corresponding tangent intersects the x-axis of the plane. We can express the tangent at the point $(\cos(\gamma), \sin(\gamma))$ as

$$\begin{aligned} t_\gamma(x) &= \begin{pmatrix} \cos(\gamma) \\ \sin(\gamma) \end{pmatrix} + x \begin{pmatrix} \cos(\gamma + \frac{3\pi}{2}) \\ \sin(\gamma + \frac{3\pi}{2}) \end{pmatrix} \\ &= \begin{pmatrix} \cos(\gamma) \\ \sin(\gamma) \end{pmatrix} + x \begin{pmatrix} \sin(\gamma) \\ -\cos(\gamma) \end{pmatrix} \end{aligned} \tag{7.24}$$

for $\gamma \in \mathbb{R}$, where $\tan(\gamma)$ is a root of $t_\gamma(x)(2) = \sin(\gamma) - x\cos(\gamma)$ for each $\gamma \neq \frac{\pi}{2} + k\pi$, $k \in \mathbb{Z}$. If the closure of real PWFA computable sets under set intersection holds, this implies that we can compute the set

$$S(Z) = \overline{\{(n\alpha + m\beta, i\alpha + j\beta, \sin(n\alpha + m\beta) - (i\alpha + j\beta)\cos(n\alpha + m\beta)) \mid i,j,m,n \in \mathbb{N}\} \cap \{(n\alpha + m\beta, i\alpha + j\beta, 0) \mid i,j,k,l \in \mathbb{N}\}}. \tag{7.25}$$

□

Remark 7.2 *Assuming that Z can be constructed in such fashion to compute a set representing the tangent, then projecting Z to the first two dimensions to obtain a PWFA computing the tangent function is a non-trivial (and possibly impossible) process, as there could exist word sequences for which the word function of Z does not converge but the word function of the projected automaton does, thus possibly introducing erroneous vectors.*

The following two problems are open, in both cases we conjecture that the answer is negative.

1. Is there a real PWFA that computes the sine, cosine or exponential function while only using rational weights?

2. Is there a real PWFA that computes the sine, cosine or exponential function on any compact set containing an infinite number of reals?

The following arguments support our conjecture that the set $\mathcal{D}(\mathbb{R})$ is not closed under the set intersection operation and that there is no PWFA that computes the exponential, sine or cosine function on the real unit interval $[0,1]$. Assume that we want to compute the exponential function e^x on the interval $[0,1]$. We conjecture that the method given in [4] is structurally the only way to achieve this using a PWFA. The x-coordinate of the result vectors is thereby computed as $n\alpha + m\beta$ for $n,m \in \mathbb{N}$ and suitably chosen α and β (see equation 7.18). If this is the only way to compute the exponential function using a PWFA, then our only way to restrict it is to restrict the set of allowed input words. A simple application of the Myhill-Nerode theorem shows that this set is not regular. A pumping lemma argument shows that is also not context free. It is thus likely that the descriptional complexity of the required language is beyond what is representable by a PWFA. Note however that the set

$$\{(n,m)|n,m \in \mathbb{N}, n\alpha + m\beta \geq 0, n\alpha + m\beta \leq 1\}$$

is decidable by a Turing machine for the choice $\alpha = 1$ and $\beta = -\sqrt{2}$, which is sufficient for the computation of the exponential function as all involved numbers are algebraic and order is decidable for real algebraic numbers (see e.g. [54]). If, on the other hand, the set of real PWFA computable sets were closed under the set intersection operation, then we could trivially compute the restriction of the exponential function to the unit interval by intersecting the sets $\{(x,e(x))|x \in \mathbb{R}\}$ and $[0,1] \times [1,e]$, which are both PWFA computable.

7.2.3 Polynomials on the real numbers

There are several ways to compute real polynomials using real PWFA, of which we will show three different approaches in this subsection. The first two constructions are similar to the construction of the automata given above for computing the exponential, sine and cosine functions and were first published in [89]. The third approach is different in two ways. Firstly it does not require the use of non-rational weights. Secondly it has a more straight-forward mapping from the set of input words to the corresponding positions on the x-axis. It was first published in [90]. The first construction is described in the following Theorem 7.7.

Theorem 7.7 *For every real polynomial $p(x) = \sum_{i=0}^{n} b_i x^i$ with $b_i \in \mathbb{R}, n \in \mathbb{N}$ the set $\{(x,p(x))|x \in \mathbb{R}\}$ is computable by a real PWFA Z that has $n+1$ states, where the first component of $f_Z(w)$ is linearly bounded by the length of w for each $w \in \Sigma_Z^*$.*

Proof: The computation of the first component of each result vector follows the scheme used in the computation of the exponential function in [4] that means for $w \in \{1,2\}^*$ the automaton computes the value $x(w) = n\alpha + m\beta$, where n is the number of 1 symbols in w and m the number of 2 symbols while $\frac{\alpha}{\beta} \notin \mathbb{Q}$ and $\alpha\beta < 0$. It suffices to show that x^k is computable while simultaneously computing x in another dimension. This is done in an inductive way. The computation of the set $\{(x,1) = (x,x^0)|x \in \mathbb{R}\}$ is done by the automaton shown in Figure 7.7. Whenever the automaton reads a symbol, it is supposed to produce either the value $p(x+\alpha)$ or $p(x+\beta)$ from the value $p(x)$ it has already produced. Without

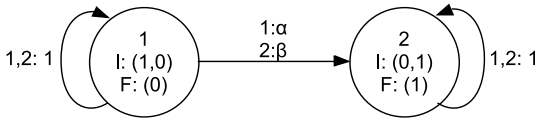


Figure 7.7: Automaton computing the set $\{(x,1) = (x,x^0) | x \in \mathbb{R}\}$ for $\alpha\beta < 0, \frac{\alpha}{\beta} \notin \mathbb{Q}$.

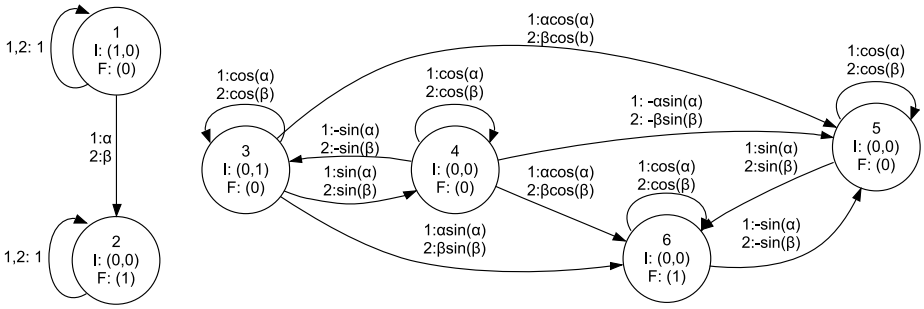


Figure 7.8: Automaton computing the function $f(x) = x\sin(x)$ by producing the set $\{(x,x\sin(x)) | x \in \mathbb{R}\}$ for $\alpha\beta < 0, \frac{\alpha}{\beta} \notin \mathbb{Q}$.

loss of generality we assume that $p(x) = x^k$ for $k > 0$ is to be evaluated at $x + \alpha$:

$$(x+\alpha)^k = \sum_{i=0}^{k} \binom{k}{i} x^{k-i}\alpha^i = x^k + \sum_{i=1}^{k} (\binom{k}{i}\alpha^i)x^{k-i} \tag{7.26}$$

By induction we already have automata for the polynomials of degrees up to $k-1$. Thus the polynomial automaton for degree k has one additional state that loops back to itself with weight 1 and has edges to the states forming a polynomial of degree $k-1$ with weights of certain binomial coefficients multiplied by some power of α that can be seen in equation 7.26. The computation of the x component can be done within the same automaton by using the linear state formed in the induction. □

As WFA over the reals are closed under addition and multiplication, multidimensional real polynomials can also be implemented by PWFA.

Example 7.2 *PWFA can compute real polynomials as well as the real sine, cosine and exponential functions using the same method for the computation of the first component. This allows us to compute products of members of these families of functions. Figure 7.8 shows an automaton that computes the function* $f(x) = x\sin(x)$ *and Figure 7.9 shows the image it computes.*

The second construction provided in Theorem 7.8 uses a slightly different way to compute the first component of the result vectors.

Theorem 7.8 *Let* $p(x) = \sum_{i=0}^{n} b_i x^i$ *be a real polynomial, where* $b_i \in \mathbb{R}, n \in \mathbb{N}$. *There is a PWFA* $Z = (Q, \Sigma = \{1,2\}, W, I, F)$ *that computes the set* $S(Z) = \{(x, p(x) | x \in \mathbb{R}\}$,

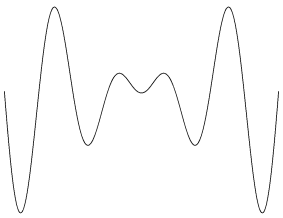


Figure 7.9: Image computed by the automaton shown in Figure 7.8 (clipped to $[-4\pi, 4\pi] \times [-4\pi, 4\pi]$).

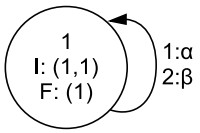


Figure 7.10: PWFA computing the set $\{(x,x)|x \in \mathbb{R}_0^+\}$ for real numbers α and β such that $0 < \alpha < 1 < \beta$ and $\frac{\ln(\alpha)}{\ln(\beta)} \notin \mathbb{Q}$.

where the first component of $f_Z(w)$ is computed as $\frac{\alpha^i}{\beta^j}$ for suitably chosen real numbers α and β, where i denotes the number of occurrences of the symbol 1 in the input word and j the number of occurrences of the symbol 2.

Proof: The set $\{\alpha^n\beta^m|n,m \in \mathbb{N}\}$ is dense in $\mathbb{R}_0^+$ if $\ln(\alpha)$ and $\ln(\beta)$ are defined, $\ln(\alpha)\ln(\beta) < 0$ and $\frac{\ln(\alpha)}{\ln(\beta)} \notin \mathbb{Q}$. This means that

$$\overline{\{(\alpha^n\beta^m, \alpha^n\beta^m)|n,m \in \mathbb{N}\}} = \left\{(x,x)|x \in \mathbb{R}_0^+\right\} \tag{7.27}$$

for suitably chosen α and β. This set is computed by the automaton shown in Figure 7.10. We can compute the function x^k on $\mathbb{R}_0^+$ as the set

$$\overline{\{(\alpha^n\beta^m, (\alpha^k)^n(\beta^k)^m)|n,m \in \mathbb{N}\}} = \{(x,x^k)|x \in \mathbb{R}_0^+\} \tag{7.28}$$

and as WFA are closed under sum and product with scalars, we can also compute

$$\overline{\{(\alpha^n\beta^m, \sum_{i=0}^{n} b_i(\alpha^i)^n(\beta^i)^m)|n,m \in \mathbb{N}\}} = \{(x,p(x))|x \in \mathbb{R}_0^+\} \ . \tag{7.29}$$

The complete polynomial can then by computed as

$$\{(x,p(x))|x \in \mathbb{R}\} = \{(x,p(x))|x \in \mathbb{R}_0^+\} \cup \{(-x,p(-x))|x \in \mathbb{R}_0^+\} \ . \tag{7.30}$$

□

The first two approaches both have in common that they use at least one weight that is not rational, even if the polynomial to be represented has only rational coefficients, so from a certain point of view they are not finite automata. We

will show now that there are PWFA computing polynomials on $\mathbb{R}$ without using non-rational weights. We start by providing a PWFA that computes a polynomial p on $\mathbb{N}$. In this approach parts of the input words can be interpreted using the functions $g[2]$ and $q[2]$, thus for the rest of the subsection we will use PWFA that have alphabets starting at 0 instead of the usual 1.

Lemma 7.5 *Let $p(x) = x^k$ with $k \in \mathbb{N}$ for $x \in \mathbb{R}$. There is a real PWFA X that computes the set $S(X) = \{(n, n^k) \,|n \in \mathbb{N}\}$.*

Proof: We construct a PWFA X' that has the alphabet $\Sigma_X = \{0,1,2\}$. Let $h(w)$ denote the word that is obtained from $w \in \Sigma_X^*$ by deleting all occurrences of the symbol 2. For each input word w we interpret the word $h(w) = b_1 b_2 \dots b_m$ as the natural number $\mathrm{nat}_2(h(w)) = \sum_{i=1}^{m} b_i 2^i$. (Note that $\mathrm{nat}_2(w)$ equals $g[2](w^R 0)$ where w^R is the reversed word of w). Let $x_1 = b_1 b_2 \dots b_m \in \{0,1\}^*$ and $x_2 = b_2 b_3 \dots b_m$ denote even natural numbers under the function nat_2. Then

$$\begin{aligned} {x_1}^k &= (2b_1 + 2x_2)^k \\ &= 2^k (b_1 + x_2)^k \\ &= 2^k \textstyle\sum_{i=0}^{k} \binom{k}{i} b_1 x_2^i \end{aligned} \tag{7.31}$$

that means

$${x_1}^k = \begin{cases} 2^k x_2^k & \text{for} \quad b_1 = 0 \\ \sum_{i=0}^{k} 2^k \binom{k}{i} x_2^i & \text{for} \quad b_1 = 1 \end{cases} . \tag{7.32}$$

So the construction of X' follows the scheme shown in Figure 7.11. The label 2 is used to give the automaton the possibility to keep the current result and produce it infinitely often, so the set computed by the automaton is not empty, as it would be, if every point were produced only once. X' produces the set

$$\{(\mathrm{nat}_2(h(w)), \mathrm{nat}_2(h(w))^k) | w \in \Sigma_X^*\} = \{(n, n^k) | n \in 2\mathbb{N}\} \tag{7.33}$$

by construction. As $(2n, (2n)^k) = (2n, 2^k n^k)$, the automaton X can be obtained by applying the invertible linear transformation $\begin{pmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{2^k} \end{pmatrix}$ to X'. □

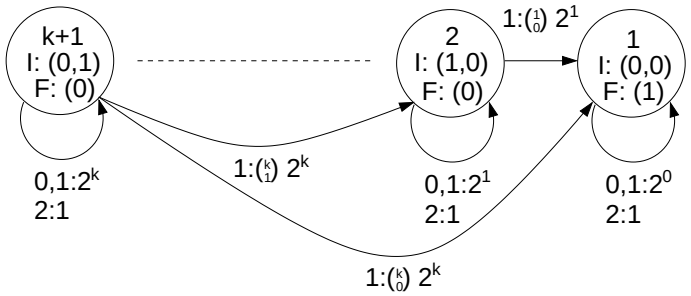


Figure 7.11: Polynomial automaton X' computing set $\{(n, n^k) | n \in 2\mathbb{N}\}$.

The proof provided for Lemma 7.5 is based on that given for the construction of WFA computing polynomials in [20]. As WFA are closed under sum and multiplication by a scalar, every real polynomial can be computed by a PWFA on $\mathbb{N}$. The construction given in Lemma 7.5 can be extended to the computation of real polynomials on $\mathbb{R}$. Firstly we show this for $\mathbb{R}_0^+$.

Lemma 7.6 *Let $p(x) = x^k$ be a real function with $k \in \mathbb{N}$. There is a PWFA Z that computes the set $\{(x, p(x)) | x \in \mathbb{R}, x \geq 0\}$ and is based on the construction given in the proof of Lemma 7.5.*

Proof: The automaton Z is constructed by combining two automata. The first is an automaton computing the set $\{x, p(x)) | x \in [0;1]\}$ by using the standard WFA construction for polynomials, let the name of this sub-automaton be Y. The second is the automaton X as defined for $p(x) = x^k$ in the proof of Lemma 7.5 but stripping away label 2 and setting its final distribution to zero. We remap X so it uses label 2 instead of 0 and 3 instead of 1. If Y makes a transition, we want X not to change its configuration, so for every state $q \in Q_X$ we introduce the edges $(q,0,q,1)$ and $(q,1,q,1)$. Both X and Y have states for x^0 to x^k in their context, let them be denoted by x_X^0 to x_X^k and x_Y^0 to x_Y^k. For every transition (x_X^i, l, x_X^j, v) where $l \in \{2,3\}$ we add a transition (x_X^i, l, x_Y^j, v). The complete scheme is shown in Figure 7.12. We observe the behavior of Z for two different classes of input

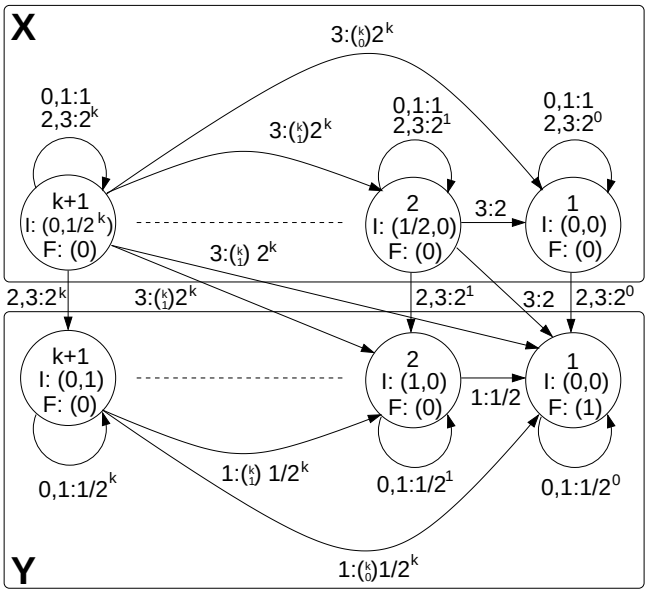


Figure 7.12: Polynomial automaton Z computing the set $S(Z) = \{x, x^k | x \in \mathbb{R}_0^+\}$.

words w:

1. $w \in (0+1)^*$ that means there is no occurrence of the symbols 2 and 3 in w. Then $f_Z(w) = (q[2](w), q[2](w)^k)$ as for usual WFA, thus $\{(x,x^k) | x \in [0;1]\} \subset S(Z)$.

2. $w \in \Sigma_Z^*(2+3)(0+1)^*$ that means there is at least one occurrence of the symbol 2 or 3 in w. As the configuration of Y is overwritten by that of X by every occurrence of the symbols 2 and 3, any occurrence of the symbols 0 and 1 before the last 2 or 3 does not have any effect on the result of the computation. Let $w' \in (2+3)^*(2+3)(0+1)^*$ be the word w with all occurrences of 0 and 1 before the last 2 or 3 erased. Let $\text{nat}_1(v) = \sum_{i=1}^{m}(a_i -$

$2)2^{i-1}$ for $v = a_1 a_2 \ldots a_m \in \{2,3\}^*$. After reading the prefix of maximal length p of w' in $(2+3)^*(2+3)$, Y is initialized to compute the set $\{(x+\mathrm{nat}_1(p), (x+\mathrm{nat}_1(p))^k) | x \in [0,1]\}$. As $\mathrm{nat}_1(p)$ can be any natural number, the computed set is

$$\bigcup_{i=0}^{\infty} \{(x+i, (x+i)^k) | x \in [0,1]\} \subset S(Z). \tag{7.34}$$

As the output for class 2 contains the output for class 1, the proof is complete. □

Again, as the set of WFA computable functions is closed under sum and multiplication by a scalar, we can obtain a PWFA computing the set $\{(x, p(x)) | x \in \mathbb{R}_0^+\}$ where p is an arbitrary real polynomial. An automaton computing the set $\{(x, p(x)) | x \in \mathbb{R}\}$ for the same polynomial can be constructed by using the set union and linear transformation constructions to build an automaton Z that computes the set

$$S(Z) = \{(x, p(x)) | x \in \mathbb{R}_0^+\} \cup \begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix} \{(x, p(-x)) | x \in \mathbb{R}_0^+\} \tag{7.35}$$

implying Theorem 7.9.

Theorem 7.9 *Let $p(x) = \sum_{k=0}^{n} b_k x^k$ be a real polynomial for $b_k \in \mathbb{R}, n \in \mathbb{N}$. There is a real PWFA Z that computes the set $\{(x, p(x)) | x \in \mathbb{R}\}$ based on the construction given in Lemma 7.6. If $b_k \in \mathbb{Q}$ for $k = 0, \ldots, n$, then the definition of Z does not require any non-rational numbers.*

7.2.4 Rational real functions

In this subsection we will show that some rational real functions are computable by real PWFA. We start with a simple lemma:

Lemma 7.7 *The sets*

$$\left\{ \left(x, \frac{a}{(bx+c)^k} \right) \middle| x \in \mathbb{R} \setminus \{y | by + c = 0\} \right\} \tag{7.36}$$

with $a, b, c \in \mathbb{R}$ are computable by real PWFA.

Proof: As mentioned above, the set $\{\alpha^n \beta^m | n, m \in \mathbb{N}\}$ is dense in $\mathbb{R}_0^+$ if $\ln(\alpha)$ and $\ln(\beta)$ are defined, $\ln(\alpha)\ln(\beta) < 0$ and $\frac{\ln(\alpha)}{\ln(\beta)} \notin \mathbb{Q}$. This means that

$$\overline{\left\{ \left(\alpha^n \beta^m, \left(\frac{1}{\alpha}\right)^n \left(\frac{1}{\beta}\right)^m \right) \middle| n, m \in \mathbb{N} \right\}} = \left\{ \left(x, \frac{1}{x} \right) \middle| x \in \mathbb{R}^+ \right\} \tag{7.37}$$

for suitably chosen α and β. The automaton shown in Figure 7.13 computes this set. Two automata computing $\{(x, 1/x) | x \in \mathbb{R}^+\}$ and $\{(-x, -1/x) | x \in \mathbb{R}^+\}$ can be merged to compute $\{x, 1/x | x \in \mathbb{R} \setminus \{0\}\}$. We can compute the set

$$\left\{ \left(x, \frac{1}{ax+b} \right) \middle| a \neq 0, x > -\frac{b}{a} \right\} \tag{7.38}$$

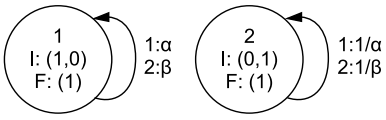


Figure 7.13: PWFA computing the set $\{(x, \frac{1}{x}) | x \in \mathbb{R}^+\}$ for real numbers α and β where $0 < \alpha < 1 < \beta$ and $\frac{\ln(\alpha)}{\ln(\beta)} \notin \mathbb{Q}$.

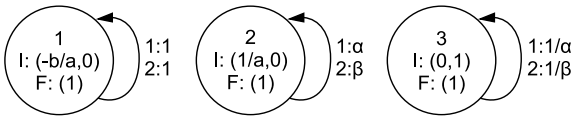


Figure 7.14: PWFA computing set $\{(x, \frac{1}{ax+b}) | a \neq 0, x > -\frac{b}{a}\}$.

by substituting $z = ax + b$ and evaluating

$$S = \left\{ \left(\frac{z}{a} - \frac{b}{a}, \frac{1}{z} \right) \middle| a \neq 0, z > 0 \right\}. \tag{7.39}$$

The automaton shown in Figure 7.14 computes the set in equation 7.39. The set described in the lemma is achieved by multiplication (in the sense of the Hadamard product of the word functions) of the second component with itself k times to obtain $\frac{1}{(bx+c)^k}$ and then adjusting the initial distribution to multiply the component by the constant a. □

So certain rational functions are computable by PWFA.

Theorem 7.10 *The set*

$$\left\{ \left(x, \frac{P(x)}{Q(x)} \right) \middle| x \in \mathbb{R} \setminus \{y | Q(y) = 0\} \right\} \tag{7.40}$$

where $P(x)$ is an arbitrary real polynomial and $Q(x) = (ax+b)^k$ for $a, b \in \mathbb{R}$ and $k \in \mathbb{N}$ is computable by a PWFA.

Proof: The statement follows from Lemma 7.7 and Theorem 7.8. □

7.2.5 Scaling functions and wavelets

It was shown in early work concerning WFA computing real functions that WFA cannot only compute smooth functions (i.e. polynomials) but also continuous non-smooth and non-continuous functions [35, 20, 34]. One class of WFA computable real functions that are not given in a closed algebraic forms are scaling functions and wavelets [32, 71, 48]. The results given in this subsection were first presented in [93].

The L^2-norm of a function $f : \mathbb{R} \mapsto \mathbb{R}$ is defined as

$$||f||_{L^2} = \left(\int_{\mathbb{R}} f(x)^2 dx \right)^{\frac{1}{2}} \tag{7.41}$$

and $L^2(\mathbb{R})$ denotes the space of square integrable functions that means $f \in L^2(\mathbb{R})$, if and only if $||f||_{L^2} < \infty$. For WFA functions in $L^2([0;1])$ i.e. functions that have support on $[0;1]$ are considered, for PWFA this restriction is not necessary. A wavelet basis of $L^2(\mathbb{R})$ is a family of functions $\psi_n^{(m)}(t)$ that are all derived from a single function called the „mother wavelet" $\psi(t)$ by translation and dilation according to

$$\psi_n^{(m)}(t) = \sqrt{2^{-m}}\psi(2^{-m}t - n) \text{ for } n,m \in \mathbb{Z} \tag{7.42}$$

such that the $\psi_n^{(m)}(t)$ are linearly independent and span $L^2(\mathbb{R})$. See for example [87] for an introduction to this topic. The scope of this discussion is limited to dyadic wavelets, where the dilation is expressed only in powers of 2. In the case of [32] these functions are not only linearly independent but also orthonormal. Wavelets are related to multi-resolution analysis [70]. One central term in this context is that of the scaling function

$$\phi(t) = \sqrt{2}\sum_{n=-\infty}^{\infty} g_0[n]\phi(2t-n), \tag{7.43}$$

where $g_0[n] \in \mathbb{R}$ is called the scaling function coefficient sequence. To be relevant for most applications, the wavelet also needs to have compact support. We will only be considering scaling functions and wavelets with finite impulse response (FIR), which means that only finitely many members of the sequence g_0 do not vanish. Given that the sequence g_0 satisfies certain properties, a mother wavelet can be built from the scaling function as

$$\psi(t) = \sqrt{2}\sum_{n=-\infty}^{\infty} g_1[n]\phi(2t-n), \tag{7.44}$$

where we call the sequence g_1 the wavelet coefficient sequence. If the sequence g_0 additionally satisfies some orthogonality constraint, then we can derive the sequence g_1 by the equation

$$g_1[n] = (-1)^{n+1}g_0[-(n-1)]. \tag{7.45}$$

The extension of the scaling function to a higher dimension is straightforward:

$$\begin{aligned}\phi(t_1,\ldots,t_d) \quad = \quad & (\sqrt{2})^d \textstyle\sum_{n_1=-\infty}^{\infty}\cdots\sum_{n_d=-\infty}^{\infty} \\ & g_0[n_1,\ldots,n_d]\phi(2t_1-n_1,\ldots,2t_d-n_d)\end{aligned} \tag{7.46}$$

We can apply the scheme used in [19] to construct a WFA that computes a dilation of the generated function on a subset of the unit hypercube of dimension d. In the example of a 2 dimensional scaling function based on 3 coefficients

for each dimension we obtain

$$\Phi_{01,01}(t_1,t_2) = \begin{cases} g(0,0)\Phi_{01,01}(2t_1,2t_2) & \text{for} \quad 0 \le t_1 < 1/2, \\ & \qquad\; 0 \le t_2 < 1/2 \\ g(1,0)\Phi_{01,01}(2t_1-1,2t_2)+ & \text{for} \quad 1/2 \le t_1 < 1, \\ g(0,0)\Phi_{12,01}(2t_1-1,2t_2) & \qquad\; 0 \le t_2 < 1/2 \\ g(0,1)\Phi_{01,01}(2t_1,2t_2-1)+ & \text{for} \quad 0 \le t_1 < 1/2, \\ g(0,0)\Phi_{01,12}(2t_1,2t_2-1) & \qquad\; 1/2 \le t_2 < 1 \\ g(1,1)\Phi_{01,01}(2t_1-1,2t_2-1)+ & \text{for} \quad 1/2 \le t_1 < 1, \\ g(1,0)\Phi_{01,12}(2t_1-1,2t_2-1)+ & \qquad\; 1/2 \le t_2 < 1 \\ g(0,1)\Phi_{12,01}(2t_1-1,2t_2-1)+ & \\ g(0,0)\Phi_{12,12}(2t_1-1,2t_2-1) & \end{cases}$$

etc. for the functions $\Phi_{01,12}, \Phi_{12,01}$ and $\Phi_{12,12}$ according to Lemma 1 in [19]. An example of a constructed WFA is given in Figure 7.15. As there is control over the coordinate axes for PWFA, this cube can be scaled to the original area of support of the scaling function. One minor problem present in [19] for WFA is that points outside the support of the function have to be computed too. This can be overcome by mapping the unused words to other sub-intervals of the support. A deeper discussion of this mapping will follow, when we present the computation of Bézier curves in Chapter 8. In certain cases there are structural similarities between automata computing scaling functions and automata computing B-splines (which we will also discuss in Chapter 8). This is not surprising, because some families of wavelets are constructed using B-splines (see e.g. [14, 87]). The 3 tap scaling function of the well-known 5/3 filter, to give an example, is exactly the uniform linear B-spline. A WFA computing this function and the graph of the function are shown in Figure 7.17. In the case of a separable transform equation 7.46 can be rewritten as

$$\begin{aligned} \phi(t_1,\ldots,t_d) \quad = \quad & (\sqrt{2})^d \textstyle\sum_{n_1=-\infty}^{\infty} \cdots \sum_{n_d=-\infty}^{\infty} \\ & g_0[n_1]\ldots g_0[n_d]\phi(2t_1-n_1,\ldots,2t_d-n_d). \end{aligned} \tag{7.47}$$

Although the examples given here are all separable, separability is not a necessary condition to build a WFA or PWFA from a scaling function of a multiresolution analysis. The original definition of the term multiresolution analysis is based on an orthonormal set of basis scaling functions. This can be relaxed to the so called biorthogonality equation

$$\left\langle \psi_n^{(m)}, \tilde{\psi}_{\tilde{n}}^{(\tilde{m})} \right\rangle = \delta[n-\tilde{n}]\delta[m-\tilde{m}] \forall n,m,\tilde{n},\tilde{m} \in \mathbb{Z}, \tag{7.48}$$

where $\tilde{\psi}$ denotes the dual wavelet of ψ and δ denotes the Kronecker δ while keeping the perfect reconstruction property. Biorthogonal transforms are popular in image compression because they, unlike orthonormal transforms (with the exception of the Haar wavelet), allow perfect reconstruction linear phase FIR transforms for two channels filterbanks resulting in dyadic decompositions of images. Linear phase filters satisfying either

$$h[d+n] = h[d-n] \; ; \; \hat{h}(\omega) = \pm|\hat{h}(\omega)|e^{-i\omega d} \tag{7.49}$$

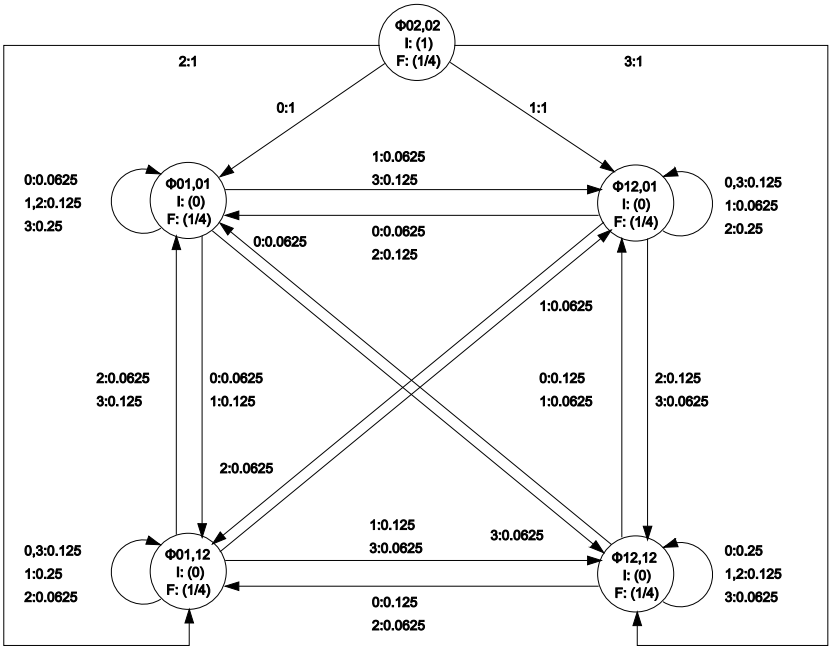


Figure 7.15: WFA computing separable scaling function of dimension two for 3 tap prototype $g_0[0] = \frac{1}{2}, g_0[1] = 1, g_0[2] = \frac{1}{2}$. The image produced by the automaton is shown in Figure 7.16.

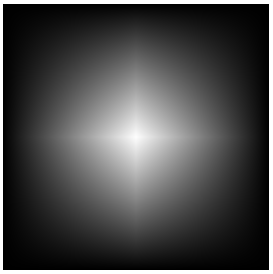


Figure 7.16: Image produced by the WFA shown in Figure 7.15.

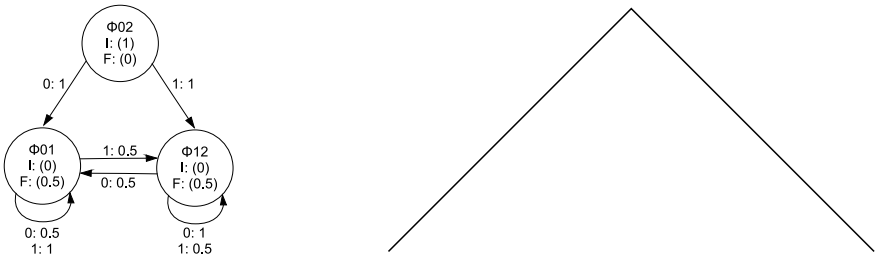


Figure 7.17: WFA computing uniform linear B-spline (left) and its curve (right).

or

$$h[d+n] = -h[d-n] \; ; \; \hat{h}(\omega) = \pm|\hat{h}(\omega)|i \times \mathrm{sgn}(\omega)e^{-i\omega d} \tag{7.50}$$

(where $\hat{h}$ denotes the Discrete Fourier Transform of h) have a symmetric or anti-symmetric impulse response relative to some center of symmetry d. Popular transforms used in image processing and compression are the 5/3 filter shown in Figure 7.15 and the 9/7 filter shown in Figure 7.18 presented in [14]. The sequences g_0 and g_1 of these filters and the sequences h_0 and h_1 of their dual filters are provided in Table 7.1 and Table 7.2. We denote a forward wavelet transform by the term wavelet analysis and a reverse transform by the term wavelet synthesis. The scaling function coefficients are provided in a form such that they satisfy the consistency condition of having a sum of 2. Note that in the case of a biorthogonal transform the coefficients g_1 are not obtained from g_0 via equation 7.45 but instead from h_0. Vice versa the sequence h_1 is obtained from g_0.

Analysis Filter Coefficients		
i	$g_0(i)$ (Scaling)	$g_1(i)$ (Wavelet)
0	1.2058980364727158	1.115087052456994
±1	0.5337282368857446	-0.5912717631142470
±2	-0.15644653305797570	-0.05754352622849957
±3	-0.03372823688574990	0.09127176311424948
±4	0.05349751482161952	
Synthesis Filter Coefficients		
i	$h_0(i)$ (Scaling)	$h_1(i)$ (Wavelet)
0	1.115087052456994	1.2058980364727158
±1	0.5912717631142470	-0.5337282368857446
±2	-0.05754352622849957	-0.15644653305797570
±3	-0.09127176311424948	0.03372823688574990
±4		0.05349751482161952

Table 7.1: Cohen, Daubechies, Feauveau 9/7 analysis and synthesis coefficients.

Analysis Filter Coefficients		
i	$g_0(i)$ (Low-pass)	$g_1(i)$ (High-pass)
0	6/4	1
±1	2/4	$-1/2$
±2	$-1/4$	
Synthesis Filter Coefficients		
i	$h_0(i)$ (Lowpass)	$h_1(i)$ (Highpass)
0	1	6/4
±1	1/2	$-2/4$
±2		$-1/4$

Table 7.2: Cohen, Daubechies, Feauveau 5/3 analysis and synthesis coefficients.

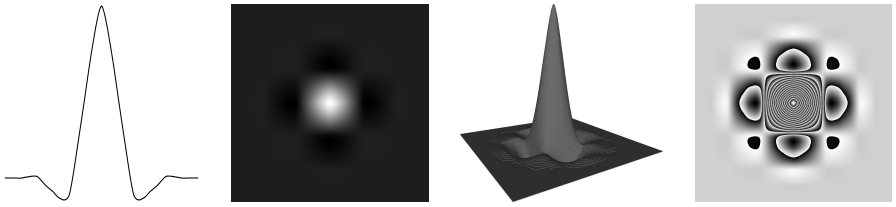


Figure 7.18: 7 tap scaling function of 9/7 filter, left to right: 1d, 2d greyscale, 3d bi-level and 2d greyscale computed with 4096 grey levels where the remainder of the division by 256 is shown. Display of the corresponding WFA having 12 (1d) respectively 57 (2d) states is omitted.

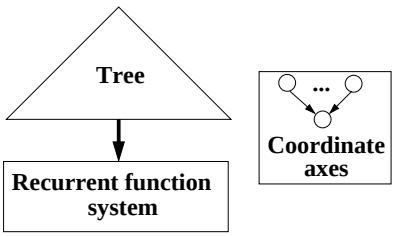


Figure 7.19: Construction scheme for PWFA computing a scaling function

The structure of a PWFA computing a scaling function can be seen in Figure 7.19. There are three parts in the automaton:

1. **Recurrent function system (RFS)**: contains the sub-hypercube functions $\Phi_{a_0,\ldots,a_{d-1}}$. It has $\prod_{i=0}^{d-1} k_i$ states, where $k_i \in \mathbb{N}$ is the maximum distance of two non-zero coefficients in component i.

2. **Tree** is a tree of order 2^d, where d is the dimension of the scaling function. The tree has enough leaves so every $\Phi_{a_0,\ldots,a_{d-1}}$ can be attached to a different outgoing edge of a leaf corresponding to its coordinates. The leaves all have the same distance from the root of the tree.

3. **Coordinate axes:** These states are used to compute the d coordinates the function depends on. The structure shown in Figure 7.19 refers to the case where the tree is complete. If the tree is not complete, this part is also prefixed with a tree of order 2^d to remap coordinates outside the support of the function to coordinates inside. The tree part is also adjusted to reflect this mapping.

The number of PWFA states S needed to compute a separable scaling function of dimension d with k nonzero coefficients is therefore in the order of magnitude

$$S = \underbrace{k^d}_{\text{RFS}} + \underbrace{\left\lceil \frac{k^d}{2^d} \right\rceil + \left\lceil \frac{\left\lceil \frac{k^d}{2^d} \right\rceil}{2^d} \right\rceil + \ldots + 1}_{\text{tree}} + \underbrace{d+1}_{\text{coord.-axes}} \tag{7.51}$$

The structure can easily be adapted to compute the corresponding wavelet. According to equation 7.44 the wavelet is just like the scaling function a linear

combination translated and dilated versions of the scaling function. All the needed components are already present in the automaton. For each $\Phi_{a_0,\ldots,a_{d-1}}$ a corresponding state $W_{a_0,\ldots,a_{d-1}}$ is inserted into the automaton and connected to the recurrent function system sub-automaton according to equation 7.44.

The dilation can be adjusted by changing the depth of the tree and the coordinate mapping. Increasing the depth of the tree by one halves the support of the represented function in every dimension it depends on. Any translation within the unit hypercube can be achieved by reassigning the edges leaving the tree leaves. The computation of a linear combination of scaling functions or wavelets of differing dilation can be done by creating a tree for the maximum depth and then adding edges with the correct weights to the leaves of the tree. If the dilation of the component corresponds to the tree's depth, this is straightforward. If it is not, still all the edges are added to leaves only, so all possible translations can be produced, because adding edges to inner nodes of the tree only allows us to add certain translations instead of all possible at the leaf level. Note that there is no need to introduce any new states, it is sufficient to add edges. Which edges and weights are needed to add a linear component not matching the depth of the tree can be computed by simulating an automaton for one that does.

If the automaton computes $\psi_0^{(0)}$ and the edges for $\psi_0^{(1)}(t) = (\sqrt{2})^{-1}\psi(2^{-1}t)$ are to be derived, this can be done by the following steps:
Let k be the depth of the tree leaves in the automaton computing ψ_0^0. For each word of length $k+2$ simulate the automaton starting at the tree root. Let $x = a_1a_2\ldots a_{k+2}$ be the current word. Then the simulation has put certain values into the states corresponding to the recurrent function system. Let this be f_i for state s_i. Then add the edges from the leaf corresponding to the word $a_2\ldots a_{k+1}$ to the states s_i with the corresponding weights f_i for label a_{k+2}.

$\mathcal{D}(\mathbb{R},d)$ is closed under set union and invertible affine transformation for each $d \in \mathbb{N}^+$ (see section 5.1), so it might be possible to save states when constructing PWFA that compute symmetric scaling functions or wavelets. We formulate the conjecture that the sub-automaton computing the recurrent function system is in fact minimal in general. This is not necessarily true for the tree part.

A nice application of wavelet linear combinations is shown in Figure 7.20. Assume that a topological map supplies height information as samples depending on longitude and latitude. Then we can build a greyscale image from this map at a certain resolution and transform this image using a given wavelet transform and build a PWFA from the given transform coefficients. Result vectors produced by decoding this PWFA can be interpreted in various ways e.g. as an approximation of the original samples of a greyscale image or as a 3d model of the landscape.

7.3 Real interval evaluation

We will show in this section that PWFA can be used to compute interval arithmetical evaluations (or shorter interval evaluations, cf. [5]) of real polynomials on the unit interval. As we interpret input words as real numbers and real intervals, we will use the input alphabet $\Sigma = \{0,1\}$.

When we compute real functions using WFA, then the input word is interpreted using the $r[2]$ function. Instead of interpreting a word $w \in \Sigma^*$ as a single

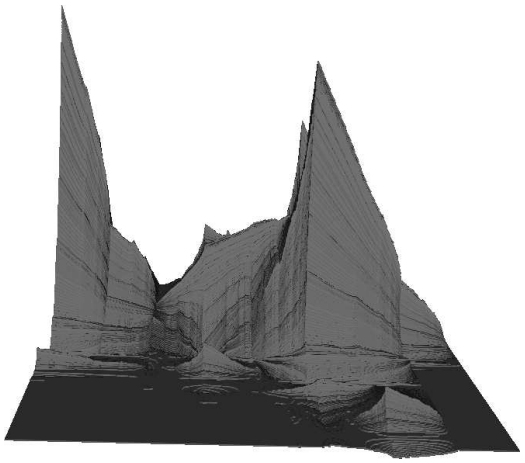


Figure 7.20: Wavelet cliffs.

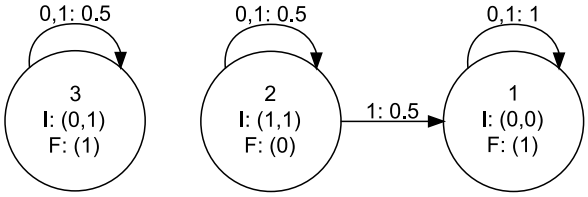


Figure 7.21: Automaton X' computing the function $f_{X'}(w)$ as defined in equation 7.53 for $w \in \{0,1\}^*$.

real number, we can also interpret it as the interval

$$r[I2](w) = \left[q[2](w), q[2](w) + 2^{-|w|}\right] \quad . \tag{7.52}$$

The interval $r[I2](w)$ contains all real numbers $r[2](w')$ such that $w' \in \{w\}\Sigma^\omega$, i.e. all extensions of w to words w' of infinite length that have the prefix w.

We start by constructing a PWFA computing the interval evaluation of the linear function $f(x) = x$.

Lemma 7.8 *There is a PWFA X that computes the function $f(w) = (\alpha,\beta)^T$ such that $[\alpha,\beta] = r[I2](w)$ for each $w \in \Sigma^*$.*

Proof: The PWFA X is shown in Figure 7.21. It computes the function

$$f_X(w) = \left(q[2](w), q[2](w) + \frac{1}{2}^{|w|}\right) \tag{7.53}$$

for each $w \in \Sigma^*$. States 1 and 2 of X describe the the linear function $f(x) = x$ for single points. State 3 is used to add the number $2^{-|w|}$ to the second component for each input word w. □

The interval computation of the linear function can be extended to general polynomials.

Theorem 7.11 *Let $p = \sum_{i=0}^{d} a_i x^i$ denote a real polynomial for some $d \in \mathbb{N}$. There is a PWFA X computing the function*

$$f_X(w) = \left(q[2](w), q[2](w) + 2^{-|w|}, a, b\right)^T \tag{7.54}$$

for each $w \in \{0,1\}^*$, where the interval $[a,b]$ is the interval evaluation of p on the interval $[q[2](w), q[2](w) + 2^{-|w|}]$.

Proof: The existence of X follows from Lemma 7.8 and the closure properties of real WFA. Some caution needs to be taken, if a_i for some $0 \leq i \leq d$ is negative, as then $a_i(q[2](w) + 2^{-|w|})^i < a_i(q[2](w))^i$ for each $w \in \{0,1\}^*$. This can be taken care of by swapping the corresponding components, when we construct the automaton corresponding to $a_i x^i$. If this issue has been considered, the rest of the construction is straight-forward. □

Let $v_1, v_2, \ldots$ be a sequence of words of increasing length in Σ^* such that v_i is a prefix of v_{i+1} for each positive natural number i. Then $\lim_{i\to\infty} 2^{-|v_i|} = 0$. Thus for each polynomial p the equations

$$\lim_{i\to\infty} q[2](v_i) = \lim_{i\to\infty} q[2]\left(v_i + 2^{-|v_i|}\right) \tag{7.55}$$

and

$$\lim_{i\to\infty} p(q[2](v_i)) = \lim_{i\to\infty} p\left(q[2]\left(v_i + 2^{-|v_i|}\right)\right) \tag{7.56}$$

hold, i.e. at infinity the intervals computed by the automaton constructed in the proof of Theorem 7.11 collapse to single points.

7.4 Complex functions

We will show that the computation of real functions given in the previous sections can be generalized to complex functions in many cases.

7.4.1 Complex polynomials on the complex unit interval

Let U denote the set given by

$$U = \{a + ib | a, b \in [0,1]\}, \tag{7.57}$$

which we call the complex unit interval. Firstly we show that every complex polynomial $p : U \mapsto \mathbb{C}$ can be computed by a complex PWFA.

Lemma 7.9 *Let a be a complex number. There is a PWFA X over $\mathbb{C}$ that computes the complex function $f(x) = x + a$ on U.*

Proof: We choose $\Sigma_X = \{1,2,3,4\}$ where we will use the aliases $(0,0)$ for the symbol 1, $(0,1)$ for 2, $(1,0)$ for 3 and $(1,1)$ for 4. Then f is computed by the automaton shown in Figure 7.22. The automaton works similar to that for the real case. In the real automaton we have two cases, either add 2^{-k} or do not add 2^{-k} for $k \geq 1$. In the complex automaton we have 4 cases, add nothing, add 2^{-k}, add $i2^{-k}$ or add $(1+i)2^{-k}$. □

As the set of complex WFA computable functions is closed under product and multiplication by a scalar and using the fundamental theorem of algebra we obtain the following theorem.

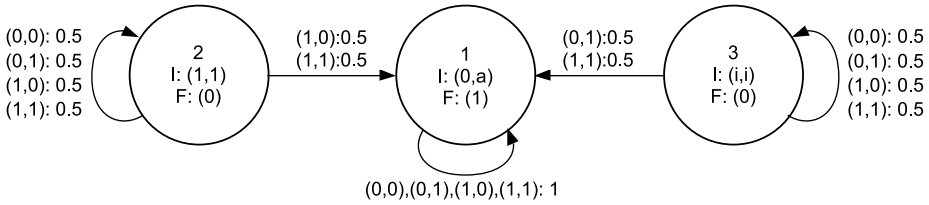


Figure 7.22: PWFA computing the complex function $f(x) = x + a$ on U for $a \in \mathbb{C}$.

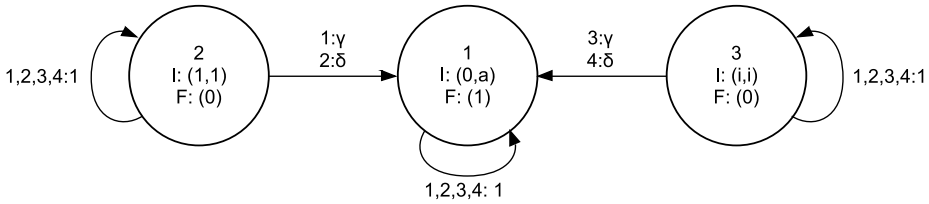


Figure 7.23: PWFA computing the complex function $f(x) = x + a$ on $\mathbb{C}$ for $a \in \mathbb{C}$, where γ, δ are real numbers such that $\gamma\delta < 0$ and $\frac{\gamma}{\delta} \notin \mathbb{Q}$.

Theorem 7.12 *Let $p(x) = \sum_{k=0}^{n} a_k x^k$ denote a complex polynomial for $n \in \mathbb{N}, a_k \in \mathbb{C}$ for $k = 0, \ldots, n$. There is a complex PWFA that computes p on U.*

As in the case of Corollary 7.2 we can generalize this result to the following corollary.

Corollary 7.3 *Let $p(x) = \sum_{k=0}^{n} a_k x^k$ denote a complex polynomial for $n \in \mathbb{N}, a_k \in \mathbb{C}$ for $k = 0, \ldots, n$. Let $s \in \mathbb{R}, s \neq 0$ and $t \in \mathbb{C}$. There is a complex PWFA that computes p on $sU + t = \{sx + t | x \in U\}$.*

7.4.2 Polynomials on the complex numbers

We can also compute complex polynomials on $\mathbb{C}$ using complex PWFA, as the following theorem shows.

Lemma 7.10 *The function $f(x) = x + a$ can be computed by a complex PWFA on $\mathbb{C}$ for each $a \in \mathbb{C}$.*

Proof: The automaton shown in Figure 7.23 computes f for real numbers γ and δ such that $\gamma\delta < 0$ and $\frac{\gamma}{\delta} \notin \mathbb{Q}$. It computes $f(x)$ as the set

$$\begin{aligned} f(x) &= \{(x, x+a) | x \in \mathbb{C}\} \\ &= \overline{\{((n\gamma + m\delta) + i(j\gamma + k\delta), (n\gamma + m\delta) + i(j\gamma + k\delta) + a) | j,k,n,m \in \mathbb{N}\}} \ . \end{aligned} \tag{7.58}$$

We can also compute f as

$$\begin{aligned} f(x) &= \{(x, x+a) | x \in \mathbb{C}\} \\ &= \overline{\{(e^{i\varphi n}\gamma^j\delta^k, e^{i\varphi n}\gamma^j\delta^k + a) | j,k,n \in \mathbb{N}\}} \ . \end{aligned} \tag{7.59}$$

for a suitable angle φ (i.e. $\frac{\varphi}{\pi} \notin \mathbb{Q}$, see the construction of the circle in [4]) and real numbers γ and δ such that $0 < \gamma < 1 < \delta$ and $\frac{\log(\gamma)}{\log(\delta)} \notin \mathbb{Q}$. A PWFA constructed for this scheme is shown in Figure 7.24.

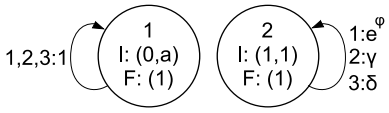


Figure 7.24: PWFA computing the complex function $f(x) = x + a$ on $\mathbb{C}$ for $a \in \mathbb{C}$, where φ is a real number such that $\frac{\varphi}{\pi} \notin \mathbb{Q}$ and γ, δ are real numbers such that $0 < \gamma < 1 < \delta$ and $\frac{\log \gamma}{\log \delta} \notin \mathbb{Q}$.

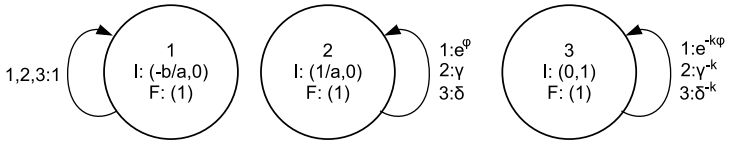


Figure 7.25: PWFA computing the set $\left\{ \left(x, \frac{a}{(bx+c)^k} \right) \middle| x \in \mathbb{C} \setminus \{y | by + c = 0\} \right\}$ with $a, b, c \in \mathbb{C}$ for an angle φ such that $\frac{\varphi}{\pi} \notin \mathbb{Q}$ and real numbers γ and δ such that $0 < \gamma < 1 < \delta$ and $\frac{\log(\gamma)}{\log(\delta)} \notin \mathbb{Q}$.

□

Again due to the closure properties of complex WFA and the fundamental theorem of algebra we obtain.

Theorem 7.13 *Let $p(x) = \sum_{k=0}^{n} a_k x^k$ denote a complex polynomial for $n \in \mathbb{N}, a_k \in \mathbb{C}$ for $k = 0, \ldots, n$. There is a complex PWFA that computes p on $\mathbb{C}$.*

7.4.3 Rational complex functions

As in the real case, the construction method for complex polynomials based on the second scheme given in the proof of Lemma 7.10 can be used to compute certain complex rational functions.

Lemma 7.11 *The sets*

$$\left\{ \left(x, \frac{a}{(bx+c)^k} \right) \middle| x \in \mathbb{C} \setminus \{y | by + c = 0\} \right\} \tag{7.60}$$

with $a, b, c \in \mathbb{C}$ are computable by complex PWFA.

Proof: Analog to the proof of Lemma 7.7, the automaton is shown in Figure 7.25. □

Theorem 7.14 *The set*

$$\left\{ \left(x, \frac{P(x)}{Q(x)} \right) \middle| x \in \mathbb{C} \setminus \{y | Q(y) = 0\} \right\} \tag{7.61}$$

where $P(x)$ is an arbitrary complex polynomial and $Q(x) = (ax + b)^k$ for $a, b \in \mathbb{C}$ and $k \in \mathbb{N}$ is computable by a PWFA.

Proof: The statement follows from Lemma 7.11 and Theorem 7.13. □

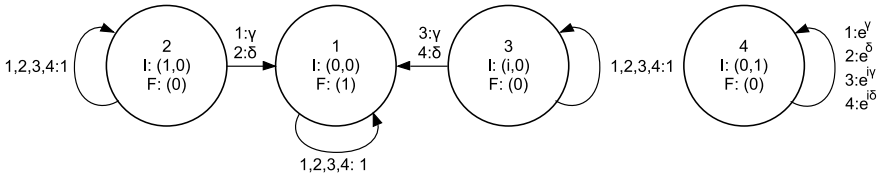


Figure 7.26: PWFA computing the function $f(x) = e^x$ on $\mathbb{C}$ for real numbers γ and δ such that $\gamma\delta < 0$ and $\frac{\gamma}{\delta} \notin \mathbb{Q}$.

7.4.4 Complex exponential function

Another complex function that can easily obtained from the real case is the complex exponential function.

Theorem 7.15 *There is a complex PWFA that computes the complex function* $f(x) = e^x$ *on* $\mathbb{C}$.

Proof: The computation of the first component was already given in the proof of Lemma 7.10. The computation of the second component again follows from the scheme presented in [4]. We can compute the complex exponential function as the set

$$\begin{aligned} f(x) &= \{(x, e^x) | x \in \mathbb{C}\} \\ &= \overline{\{((n\gamma + m\delta) + i(j\gamma + k\delta), e^{(n\gamma+m\delta)+i(j\gamma+k\delta))} | j,k,n,m \in \mathbb{N}\}} \\ &= \overline{\{((n\gamma + m\delta) + i(j\gamma + k\delta), e^{n\gamma} e^{m\delta} e^{ij\gamma} e^{ik\delta} | j,k,n,m \in \mathbb{N}\}} \end{aligned} \tag{7.62}$$

for $\gamma\delta < 0, \frac{\gamma}{\delta} \notin \mathbb{Q}$. The automaton is shown in Figure 7.26. □

Chapter 8

Splines

PWFA are able to represent polynomials in a very compact form. One example is the automaton shown in Figure 4.2 that computes a parametric curve based on two polynomials. Splines are a family of functions that are piece-wise defined by polynomials. As the set $\mathcal{D}(\mathbb{R},d)$ is effectively closed under set union and invertible affine transformation for each $d \in \mathbb{N}^+$, piece-wise defined polynomials can be represented easily using PWFA. We will show that three types of spline defined curves and relations can be computed by PWFA. These are Bézier curves, Catmull-Rom splines and B-splines. A good overview over splines is given in [41]. A more general introduction to 2d computer graphics can be found in [79]. A deeper discussion of the Catmull-Rom splines used for image magnification can be found in [13], simple introductions to the topic can be found in many places (see [7, 37, 61]). The construction of Bézier curves is based on the well-known Bernstein polynomials. A thorough discussion of B-splines can be found in [33]. Most constructions of PWFA computing spline curves given in this chapter were first presented in [88].

8.1 Bernstein polynomials

An introduction to Bernstein polynomials can be found in many textbooks. The introduction given here is based on [41]. The functions are mainly considered on the unit interval $[0,1]$ in applications. The original purpose of the invention of these functions by Bernstein was to use them in a formulation of a constructive proof of the Weierstraß approximation theorem. The theorem states that each continuous curve $c(t)$ defined on $[0,1]$ can be approximated with arbitrary precision using polynomials. This result is however of mostly theoretical value, as an increase of approximation quality requires the use of polynomials of ever higher degree, which is not suitable in applications, as it implies numerical problems and high computational cost.

A Bernstein polynomial $B_i^n(t)$ is given by

$$B_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i} \tag{8.1}$$

where $\binom{n}{i}$ is defined as

$$\binom{n}{i} = \begin{cases} \frac{n!}{i!(n-i)!} & \text{if } 0 \le i \le n \\ 0 & \text{otherwise.} \end{cases} \tag{8.2}$$

Bernstein polynomials can be written recursively as

$$B_i^n(t) = (1-t)B_i^{n-1}(t) + tB_{i-1}^{n-1}(t) \tag{8.3}$$

where

$$B_0^0(t) \equiv 1 \tag{8.4}$$

and

$$B_j^n(t) \equiv 0 \text{ for } j \notin \{0,\ldots,n\}. \tag{8.5}$$

A practically important property is that the Bernstein polynomials for a chosen natural number n always sum up to 1 for each t, i.e.

$$\sum_{j=0}^{n} B_j^n(t) \equiv 1 \quad . \tag{8.6}$$

Another property is that the Bernstein polynomials B_i^n are linear independent for each n, i.e. they form a basis of the linear space of all polynomials of degree n.

Figure 8.1 shows the Bernstein polynomials for the case of $n = 3$, which is the most common in applications.

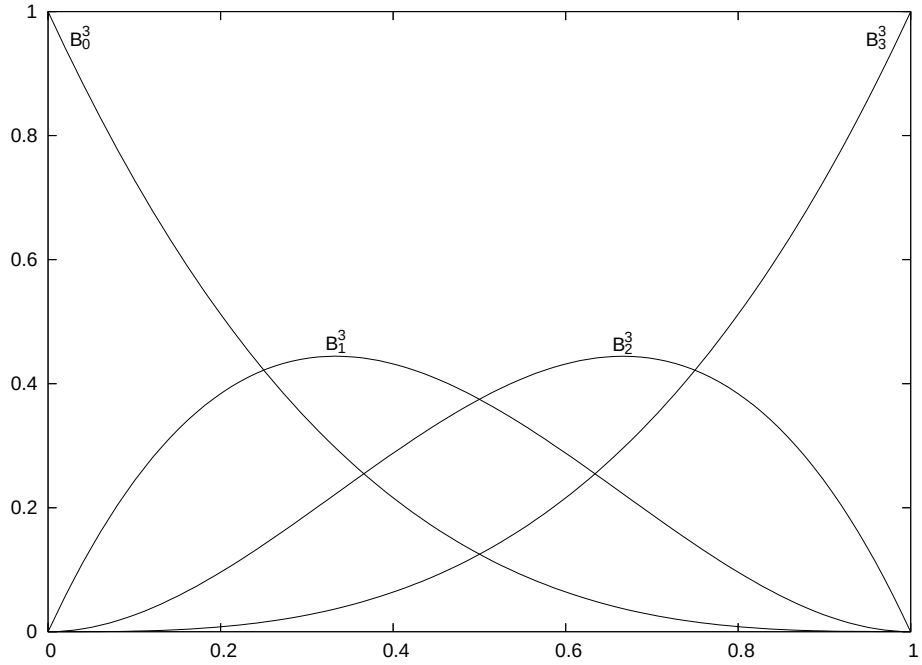


Figure 8.1: Bernstein polynomials B_0^3 to B_3^3 plotted on the unit interval.

8.2 Bézier curves

Bézier curves where developed by Pierre Bézier for automobile design in the 1960s (cf. [41]). They are based on Bernstein polynomials. A Bézier curve of order n is a real function given by the equation

$$B^n(t) = \sum_{i=0}^{n} B_i^n(t) P_i, \tag{8.7}$$

where the P_i are real numbers, which are called the control points of the curve. Each curve $B^n(t)$ is a real polynomial considered on the unit interval $[0,1]$. Thus it is very easy to construct a PWFA computing a Bézier curve. The control point sequence is often written as a vector, i.e. $P_i = (P_0, P_1, \ldots, P_n)$ (where we separate the elements by commas to improve readability).

Example 8.1 *Let $n = 3$ and $P_i = (0, 0, -\frac{1}{3}, 0)$. Then the polynomial we obtain is*

$$\begin{aligned} p(t) &= -\tfrac{1}{3} B_2^3 \\ &= -\tfrac{1}{3}(3t^2 - 3t^3) \\ &= t^3 - t^2 \ . \end{aligned} \tag{8.8}$$

Some important properties of Bézier curves are (according to [41]):

- Symmetry: the control point vectors $(P_0, \ldots, P_n)$ and $(P_n, \ldots, P_0)$ describe the same curve. It is however traversed in opposite directions in the two cases.
- Endpoint interpolation: The curve interpolates (touches) the endpoints, i.e. for each $n \in \mathbb{N}$ and $P = (P_0, \ldots, P_n)$ the equations

  $$\sum_{i=0}^{n} B_i^n(0) P_i = P_0 \tag{8.9}$$

  and

  $$\sum_{i=0}^{n} B_i^n(1) P_i = P_n \tag{8.10}$$

  hold.
- Convex hull property: the curve is inside the convex hull of the control points.
- Pseudo-local control: Each Bernstein polynomial B_i^n has only one maximum which it assumes at the point $t = i/n$. It influences the Bézier curve mainly in the area of this maximum. Points that lay relatively far away are only influenced slightly, if P_i is changed.

The derivative of a Bézier curve is given by

$$\frac{d}{dt} B^n(t) = n \sum_{i=0}^{n-1} (P_{i+1} - P_i) B_j^{n-1}(t) \tag{8.11}$$

for $n \in \mathbb{N}, n > 0$ (see [41]).

A Bézier curve $B^n(t)$ of degree n can be split into two Bézier curves $B^{n\prime}(t)$ and $B^{n\prime\prime}(t)$ of degree n at each point $t_0 \in (0,1)$ such that

$$B^n(t) = \begin{cases} B^{n\prime}(\frac{t}{t_0}) & \text{for } t \in [0,t_0) \\ B^{n\prime\prime}(\frac{t-t_0}{1-t_0}) & \text{for } t \in [t_0,1] \end{cases} . \tag{8.12}$$

This property can be used to draw approximations of Bézier curves with arbitrary precision. An initial curve is split into halves until each obtained sub-curve sufficiently resembles a straight line. These straight lines are then rendered. This drawing algorithm for Bézier curves is called the *De Casteljau algorithm* (cf. [41]).

Parametric curves can be obtained by substituting the real numbers found in a control point vector by vectors taken from a higher dimensional real space. A parametric 2d curve with $n = 3$ is e.g. given by a control point vector of the type $P = ((x_0,y_0),(x_1,y_1),(x_2,y_2),(x_3,y_3))$.

In applications we mostly encounter Bézier curves where $n = 3$, which are called *cubic Bézier curves*. These curves have many applications, ranging from the description of font outlines to the modeling of 3d objects. Cubic Bézier curves have an additional very intuitive property describing how the two non end-points influence the represented curve. If we evaluate the derivative of a cubic Bézier curve at the points $t = 0$ and $t = 1$, we see that the tangent of the curve at P_0 points to P_1 and the tangent at P_3 points to P_2. This property makes it very simple to check, whether a given cubic Bézier curve is similar to a straight line. An example is again given by the automaton shown in Figure 4.2.

Example 8.2 *Consider the control point vector* $P = ((0,0),(0,-\frac{1}{3}),(-\frac{1}{3},-\frac{1}{3}),(0,0))$*, for which we obtain the parametric curve* $B(t) = (t^3-t^2,t^2-t)$*. The corresponding curve is shown in Figure 8.2. The curve starts and ends at the origin. The two tangents intersect the points* $(0,-\frac{1}{3})$ *and* $(-\frac{1}{3},-\frac{1}{3})$ *respectively.*

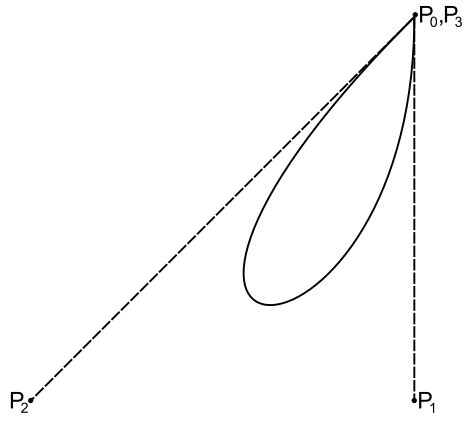


Figure 8.2: Parametric cubic Bézier curve $B(t) = (t^3-t^2,t^2-t)$ defined by the control point vector $P = ((0,0),(0,-\frac{1}{3}),(-\frac{1}{3},-\frac{1}{3}),(0,0))$ plotted for $t \in [0,1]$.

Another example is the approximation of circles and arcs by cubic Bézier curves, as it is often used in fonts.

Example 8.3 *Assume that we want to approximate the top right quarter of a circle of unit radius using a cubic Bézier curve. The choice of the points* $P_0 = (1,0)$ *and* $P_3 = (0,1)$ *is straightforward, as we want the curve to interpolate these points. It is also clear that the tangent at* P_0 *should be vertical and the tangent at* P_3 *should be horizontal. This implies that* $P_1 = (1,k_1)$ *and* $P_2 = (k_2,1)$ *for some real numbers* k_1 *and* k_2*. For reasons of symmetry it is obvious that* $k_1 = k_2 = k$*, so* $P = ((1,0),(1,k),(k,1),(0,1)$*. Thus we have to find a number* k *such that the curve is an optimal approximation of the circle sector. The derivation of the number* k *can be found in literature (see [69, 86]). The optimal value is*

$$k = 4\frac{\sqrt{2}-1}{3} \approx 0.552 \ . \tag{8.13}$$

In practice a curve is usually not approximated by a single Bézier curve. Instead a curve is commonly first partitioned into segments and then each segment is approximated by a separate curve that is shifted into the right position (remember that all Bézier curves are defined on $[0,1]$). This can be easily done with PWFA, because we have effective algorithms for the set union and invertible affine transformation operations. In fact, as all curve segments are built from polynomials, we can obtain very compact automata. Assume that we want to approximate a continuous curve $c(t)$ using a PWFA that is defined on the interval $[0,2^q]$ for some natural number q. We partition the curve into a set of curves $c_i(t)$ such that

$$c_i(t) = \begin{cases} c(t+i) & \text{for } 0 \le t < 1 \text{ and } i < 2^q - 1 \\ c(t+i) & \text{for } 0 \le t \le 1 \text{ and } i = 2^q - 1 \\ 0 & \text{otherwise} \end{cases} \tag{8.14}$$

for $i = 0, \ldots, 2^q - 1$. Then we apparently obtain

$$c(t) = \sum_{i=0}^{2^q-1} c_i(t-i) \ . \tag{8.15}$$

Thus to represent the complete curve, we can use the construction

$$\begin{array}{rcl} c & = & \{(t, \sum_{i=0}^{2^q-1} c_i(t-i)) | t \in [0, 2^q - 1]\} \\ & = & \bigcup_{i=0}^{2^q-2} \{(t+i, c_i(t)) | t \in [0,1)\} \cup \{(t + 2^q - 1, c_{2^q-1}(t)) | t \in [0,1]\} \\ & = & \bigcup_{i=0}^{2^q-1} \{(t+i, c_i(t)) | t \in [0,1]\} \end{array} \tag{8.16}$$

where the last step is valid because c is continuous. Now assume that we have approximated each curve segment $c_i(t)$ by a Bézier curves of order n, i.e.

$$\begin{array}{rcl} c_i(t) & \approx & p_i(t) \\ & = & \sum_{j=0}^{n} P_{i,j} B_j^n(t) \\ & = & \sum_{j=0}^{n} P'_{i,j} x^j \end{array} \tag{8.17}$$

for $i = 0, \ldots, 2^q - 1$ such that

$$p(t) = \sum_{i=0}^{2^q-1} p_i(t-i) \tag{8.18}$$

is a continuous function. Then we can construct a PWFA X that computes $p(t)$ as the set

$$p = \bigcup_{i=0}^{2^q-1} \{(t+i, p_i(t)) | t \in [0,1]\} \tag{8.19}$$

that uses

$$|Q_X| = 2\left(\frac{q}{2} + \frac{q}{4} + \ldots + 1\right) + \max\{(n+1), 2\} \tag{8.20}$$

states. X uses $\max\{n+1, 2\}$ states to represent a WFA defined polynomial. This maximum will usually be $n+1$. For sake of completeness, we have also included the case of $n = 0$ (i.e. approximation by piece-wise constant functions). If $n = 0$ we have to introduce an additional state, because we require a linear function for the computation of the first component of the result vectors. The geometric sum stems from a set union construction which yields a tree structured automaton for each of the two components of the result vectors. If the number of sections is not a power of 2, then we can substitute non-existent intervals by repeating existing intervals. E.g. assume that we want to represent the curve $c(t)$ defined on $[0,3]$ and have decomposed it as

$$c(t) = c_0(t-0) + c_1(t-1) + c_2(t-2) \tag{8.21}$$

which we compute as

$$c = \{(t+0, c_0(t)) | t \in [0,1]\} \cup \{(t+1, c_1(t)) | t \in [0,1]\} \cup \{(t+2, c_2(t)) | t \in [0,1]\} \ . \tag{8.22}$$

Then we can repeat the interval $[0,1]$ for the non-existing interval $[3,4]$ by using

$$\begin{aligned} c \quad = \quad & \underbrace{\{(t+0, c_0(t)) | t \in [0,1]\}}_{\text{prefix 11}} \cup \underbrace{\{(t+1, c_1(t)) | t \in [0,1]\}}_{\text{prefix 12}} \cup \\ & \underbrace{\{(t+2, c_2(t)) | t \in [0,1]\}}_{\text{prefix 21}} \cup \underbrace{\{(t+0, c_0(t)) | t \in [0,1]\}}_{\text{prefix 22}} \ . \end{aligned} \tag{8.23}$$

Example 8.4 *Figure 8.3 shows an automaton computing a curve $c(t)$ defined on $[0,3]$ built from 3 Bézier curves. The image computed by the automaton is shown in Figure 8.4. The control point vectors and corresponding polynomials are*

- $(0, 0, \frac{11}{3}, 4) \cong -7x^3 + 11x^2$ *for* $[0,1]$,
- $(4, \frac{13}{3}, \frac{13}{6}, 2) \cong 4.5x^3 - 7.5x^2 + x + 4$ *for* $[1,2]$ *and*
- $(2, \frac{11}{6}, 3, 3) \cong -2.5x^3 + 4x^2 - 0.5x + 2$ *for* $[2,3]$.

(Remember that the polynomials are all evaluated on $[0,1]$ and then shifted into their respective position). The polynomial part of the automaton is contained in the states 1 to 4. States 5, 6 and 7 are the union tree for the first component. States 8, 9 and 10 are the union tree for the second component. The non-existent interval $[3,4]$ has been substituted by the interval $[0,1]$ in the tree leafs, i.e. the configuration of the automaton after reading the word 22 is the same as after reading the word 11.

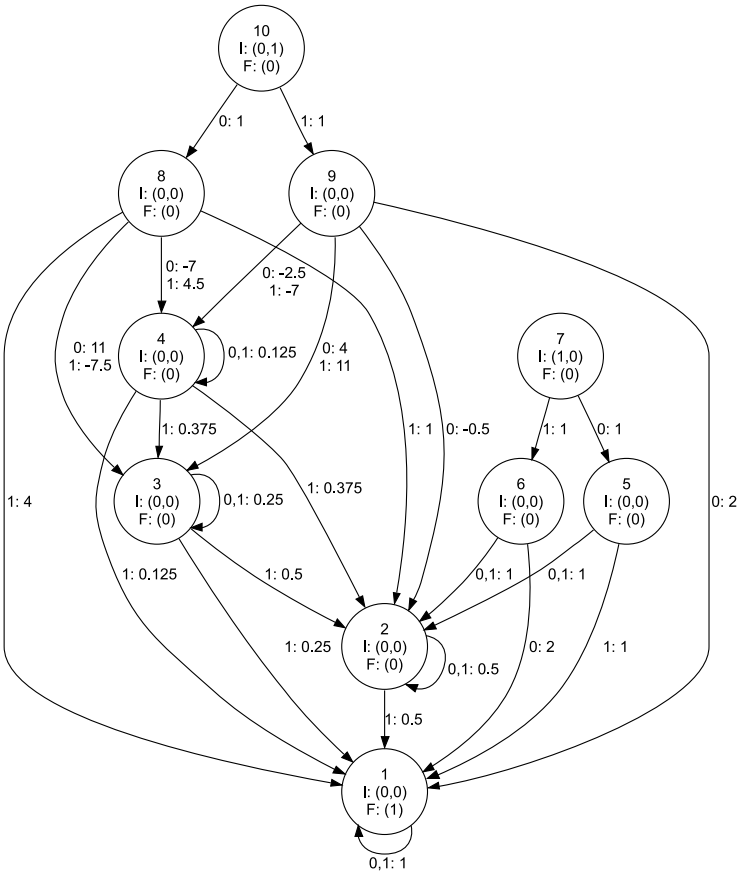


Figure 8.3: Automaton computing a curve built from 3 Bézier curves.

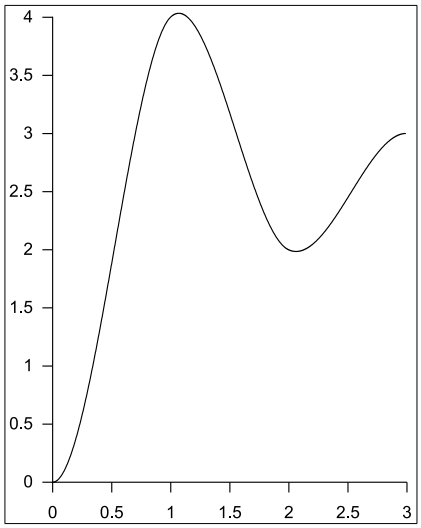


Figure 8.4: Function computed on $[0,3]$ by the automaton shown in Figure 8.3.

If the number of sections is not a power of two, then another option is to replace the complete binary trees used above with incomplete binary trees. This however means that different word prefixes may correspond to different stages of computation progress.

We have restricted our attention to continuous curves above to keep notations simple and for being able to express a curve as a sum of shifted curves defined on the unit interval. The construction of an automaton from a curve does however work in the same way for curves that are not continuous (at a finite number of points), as the summing is expressed using a set union construction, which works on arbitrary relations instead of functions. If we consider a non-continuous curve, then the computed set will not describe a function. This is due to the topological closure operation performed during PWFA computation. Consider e.g. the function

$$c(t) = \begin{cases} 1 & \text{if } 0 \leq t \leq 1 \\ 0 & \text{otherwise,} \end{cases} \tag{8.24}$$

which we call a unit pulse. Then a PWFA constructed to compute this curve as described above will also produce the vectors $(0\ 0)^T$ and $(1\ 0)^T$. Thus the automaton actually produces the wrong set. This is however irrelevant in applications, as the error only occurs as a consequence of a topological closure operation.

8.3 Bézier patches

Bézier curves can also be extended to the representation of surfaces, which are called Bézier patches. For sake of simplicity, we will limit the discussion to cubic surfaces parametric in two (real) variables, which are called bicubic Bézier patches. The extensions to more variables and higher degree is straightforward. A bicubic Bézier patch is defined by a matrix Q of $4 \cdot 4 = 16$ coefficients and the formula

$$B^3(u,v) = \sum_{i=0}^{3} \sum_{j=0}^{3} Q(i,j) B_i^3(u) B_j^3(v) \ . \tag{8.25}$$

Bézier patches inherit some of the properties of Bézier curves. The corner points corresponding to the argument pairs $(0,0)$, $(0,1)$, $(1,0)$ and $(1,1)$ are interpolated, e.g. the patch assumes the value $Q(0,0)$ for the argument pair $(0,0)$. If we fix the parameter u to a certain value in $[0,1]$, then we obtain a Bézier curve in v and vice versa. In particular, the border lines of the patch are defined by Bézier curves.

Bézier patches can be represented by PWFA. The construction can easily be derived from the results given in Theorem 7.1 and Corollary 7.1.

The gradient at point (u,v) is as usual obtained by partial differentiation of the patch function, i.e.

$$\nabla B^3(u,v) = \begin{pmatrix} \frac{\partial B^3(u,v)}{\partial u} \\ \frac{\partial B^3(u,v)}{\partial v} \end{pmatrix} \ . \tag{8.26}$$

If the elements of the matrix Q are 3 dimensional real vectors, a Bézier patch describes a surface in $\mathbb{R}^3$. Such surfaces can be used for 3d modeling.

The normal vectors required for the computation of lighting effects in computer graphics (see [96]) can be computed using partial derivatives and the cross product as

$$N(u,v) = \frac{\partial B^3(u,v)}{\partial u} \times \frac{\partial B^3(u,v)}{\partial v}, \tag{8.27}$$

where $\times$ denotes the cross product. As the partial derivatives are still polynomials and the cross-product produces a vector of three polynomials in u and v, the function $N(u,v)$ is also computable by a PWFA. In applications it may however be necessary to interpolate certain values from the neighborhood of a point, as the function $N(u,v)$ may compute the zero vector. The zero vector however does not define a direction in $\mathbb{R}^3$, as it is required for the computation of lighting effects.

The following example shows that we can use cubic Bézier patches to approximate spheres in 3 dimensional real space.

Example 8.5 *We can extend the curve given in Example 8.3 to the patch case. We construct an approximation of the octant of a sphere with unit radius that has only non-negative coordinate elements (in a right handed coordinate system where the middle finger denotes the z-axis, the index finger the y-axis and the thumb the x-axis, this is the one situated in the right (x) - top (y) - front (z) octant). We obtain the border coefficients of the matrix Q directly from the curve case, thus we have:*

$$Q = \begin{pmatrix} (0\ 1\ 0) & (k\ 1\ 0) & (1\ k\ 0) & (1\ 0\ 0) \\ (0\ 1\ 0) & a & b & (1\ 0\ k) \\ (0\ 1\ 0) & c & d & (k\ 0\ 1) \\ (0\ 1\ 0) & (0\ 1\ k) & (0\ k\ 1) & (0\ 0\ 1) \end{pmatrix} \tag{8.28}$$

where k is chosen as above and a,b,c and d are vectors in $\mathbb{R}^3$. We do not provide optimal values for a,b,c and d. However, we regard the choice of

$$a = (k\ 1\ 0) + \sin\left(\frac{\pi}{6}\right)(0\ 0\ k) \tag{8.29}$$

$$b = (1\ k\ 0) + \sin\left(\frac{\pi}{3}\right)(0\ 0\ k) \tag{8.30}$$

$$c = (0\ 1\ k) + \sin\left(\frac{\pi}{6}\right)(k\ 0\ 0) \tag{8.31}$$

$$d = (0\ k\ 1) + \sin\left(\frac{\pi}{3}\right)(k\ 0\ 0) \tag{8.32}$$

as a good approximation.

Let S be the unit sphere (i.e. the set $S = \{(x,y,z) | x,y,z \in \mathbb{R}, |x^2+y^2+z^2| = 1\}$). Then we can construct a PWFA X of dimension 3 over $\mathbb{R}$ such that $S(X) = S$. One such automaton X can be obtained by first constructing an automaton X' computing a circle in the x,y plane of $\mathbb{R}^3$ and then applying an iterated affine transformation representing a rotation by an angle of $\arccos(4/5)$ around the x-axis to X'. The use of an approximation however allows us to texture the Bézier patch using a WFA coded image, as the interpretation of the input words corresponds to the texture mapping found 3d computer graphics. This means that we identify the corners of the WFA coded image with coordinates in $[0,1]^2$ with the corner points of the matrix Q used to define the patch. An example thereof is shown in Plate I in the appendix.

The interval evaluation of real polynomials on the unit interval using PWFA presented in section 7.3 can be extended to the computation of Bézier patches. Assume that we are given a bicubic Bézier patch by the control points $Q_{i,j}$ for $(i,j) \in \{0,\ldots,3\}^2$. The patch is then defined by equation 8.25. Let $w = a_1 \ldots a_n \in \{0,1,2,3\}^*$ for some natural number n. Furthermore, let the functions $g_1(x) : \{0,1,2,3\} \mapsto \{0,1\}$ and $g_2(x) : \{0,1,2,3\} \mapsto \{0,1\}$ be defined by $g_1(a) = a \bmod 2$ for $a \in \{0,1,2,3\}$ and $g_2(a) = \lfloor \frac{a}{2} \rfloor$ for $a \in \{0,1,2,3\}$. Then instead of the single point $B^3(q[2](g_1(a_1)\ldots g_1(a_n)), q[2](g_2(a_1)\ldots g_2(a_n)))$ we compute the four points

$$p_{0,0} = B^3(q[2](g_1(a_1)\ldots g_1(a_n)), q[2](g_2(a_1)\ldots g_2(a_n))), \tag{8.33}$$

$$p_{1,0} = B^3(q[2](g_1(a_1)\ldots g_1(a_n)) + 2^{-n}, q[2](g_2(a_1)\ldots g_2(b_n))), \tag{8.34}$$

$$p_{0,1} = B^3(q[2](g_1(a_1)\ldots g_1(a_n)), q[2](g_2(a_1)\ldots g_2(a_n)) + 2^{-n}) \text{ and} \tag{8.35}$$

$$p_{1,1} = B^3(q[2](g_1(a_1)\ldots g_1(a_n)) + 2^{-n}, q[2](g_2(a_1)\ldots g_2(a_n)) + 2^{-n}), \tag{8.36}$$

which mark the corners of a quadrangle $Q_n(w) = (p_{0,0}, p_{0,1}, p_{1,1}, p_{1,0})$ with the edges $(p_{0,0}, p_{0,1})$, $(p_{0,1}, p_{1,1})$, $(p_{1,1}, p_{1,0})$ and $(p_{1,0}, p_{0,0})$. We call the obtained automaton the quadrangle PWFA of the patch. Let Q_n denote the set of all such quadrangles for word length n, i.e.

$$Q_n = \bigcup_{w \in \Sigma^n} Q_n(w) \ . \tag{8.37}$$

The sequence $(Q_n)_{n=0}^{\infty}$ converges to some set of quadrangles Q_∞. All quadrangles in Q_∞ denote single points and the set of all these points is exactly the set S of points on the Bézier patch, which is given by

$$S = \{p | p = B^3(u,v) \text{ for some } (u,v) \in [0,1]^2\} \ . \tag{8.38}$$

Example 8.6 *Example 8.5 describes a bicubic Bézier patch representing one octant of a sphere. Let X denote the quadrangle PWFA for this patch. Figure 8.5 shows depictions of the sets Q_0 to Q_7, which are computed by X for the word lengths 0 to 7 respectively. The set Q_i contains 4^i quadrangles for each $i \in \mathbb{N}$.*

In addition to the four corner points computed for a rectangle, we can also compute a corresponding normal vector for each corner point by increasing the dimension of a corresponding quadrangle PWFA. The normal vectors can be computed by generalizing equation 8.27 to the quadrangle case.

8.4 Catmull-Rom splines

Catmull-Rom splines were presented in [13] and are a special case of Hermite cubic curves. They are very popular in image resampling. A Catmull-Rom spline defined by the control point vector $P_i = (P_0, \ldots, P_n)$ interpolates each point P_j for $j = 0, \ldots, n$. This is why Catmull-Rom splines are called interpolating splines in contrast to approximating splines.

A Hermite cubic curve is constructed from 4 constraints. It is supposed to start at a value p_0, where the interpolating curve has gradient m_0 and end at the

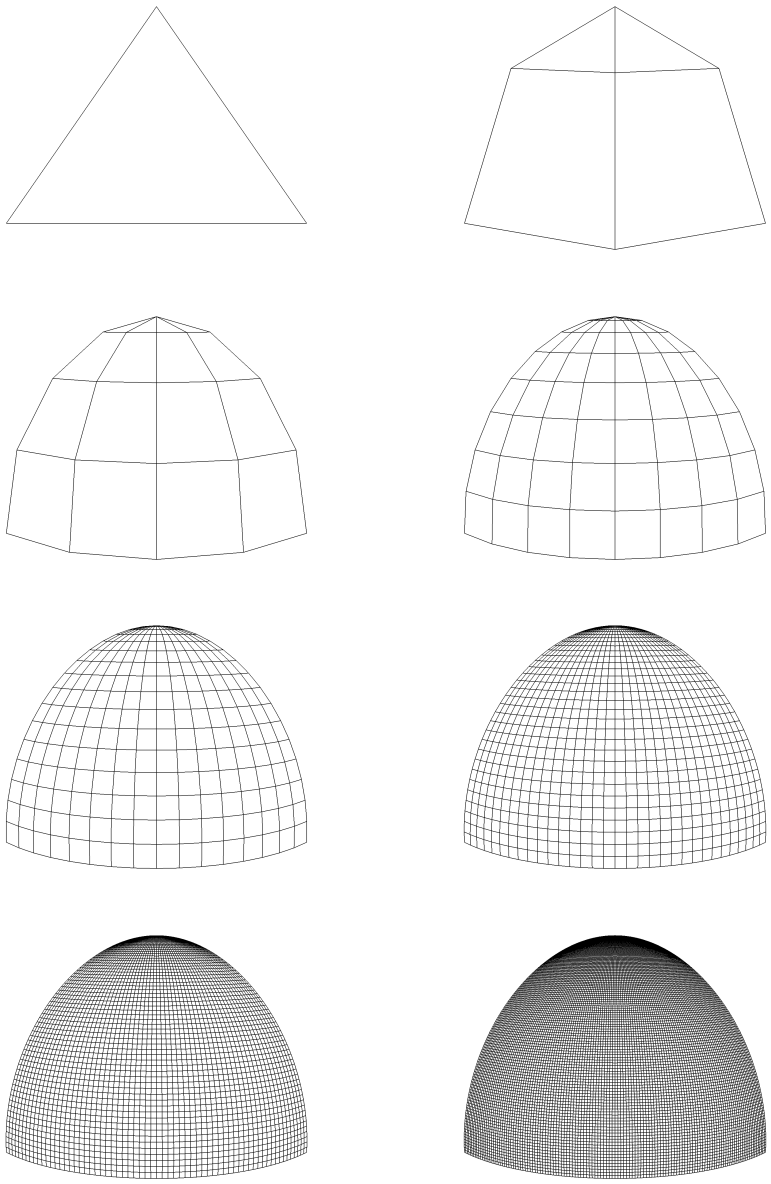


Figure 8.5: Display of the sets Q_0 to Q_7 computed by the quadrangle PWFA obtained from the bicubic Bézier patch given in Example 8.5. Note that each set contains quadrangles only. In the special case of this patch, some quadrangles degenerate to triangles, as two of their four corner points are identical.

value p_1, where the gradient of the curve is m_1. Written as a set of formulae, this means that the curve $c(t)$ (again considered on $[0,1]$) has the properties

$$c(0) = p_0,\ c(1) = p_1,\ \frac{dc}{dt}(0) = m_0,\ \frac{dc}{dt}(1) = m_1. \tag{8.39}$$

We have seen above that the Bernstein polynomials of degree n form a basis of the linear space of polynomials of degree n. Thus we can construct the curve $c(t)$ as a cubic Bézier curve, where we follow the derivation given in [41]. A Bézier curve interpolates its endpoints, thus it is clear that we have $P_0 = p_0$ and $P_3 = p_1$. The derivatives of a cubic Bézier curve at its endpoints are given by the equations

$$\frac{d}{dt}B^3(0) = 3(P_1 - P_0) \tag{8.40}$$

and

$$\frac{d}{dt}B^3(1) = 3(P_3 - P_2)\ . \tag{8.41}$$

Thus we obtain

$$P_1 = p_0 + \frac{1}{3}m_0 \tag{8.42}$$

and

$$P_2 = p_1 - \frac{1}{3}m_1 \tag{8.43}$$

and the curve can be written as

$$c(t) = p_0 B_0^3(t) + \left(p_0 + \frac{1}{3}m_0\right) B_1^3(t) + \left(p_1 - \frac{1}{3}m_1\right) B_2^3(t) + p_1 B_3^3(t) \tag{8.44}$$

where the control point vector of the curve is $(p_0, (p_0 + \frac{1}{3}m_0), (p_1 - \frac{1}{3}m_1), p_1)$. A Hermite cubic curve is usually written in its *cardinal form*. A cardinal form of a (spline) curve is one, where the controlling entities explicitly and solely appear in the formulation. This form is given by

$$c(t) = p_0 H_0^3(t) + m_0 H_1^3(t) + m_1 H_2^3(t) + p_1 H_3^3(t) \tag{8.45}$$

for the Hermite cubic curve where the basis functions $H_0^3(t)$ to $H_3^3(t)$ are easily computed as

$$\begin{array}{lcl} H_0^3(t) & = & B_0^3(t) + B_1^3(t) \\ H_1^3(t) & = & \frac{1}{3}B_1^3(t) \\ H_2^3(t) & = & -\frac{1}{3}B_2^3(t) \\ H_3^3(t) & = & B_2^3(t) + B_3^3(t)\ . \end{array} \tag{8.46}$$

We have now seen that each instance of a Hermite cubic curve can be represented as a Bézier cubic curve. Thus it is clear that we can compute single Hermite cubic curves and curves that can be decomposed into shifted Hermite cubic curves using PWFA, i.e. we can build splines from Hermite cubic curves.

The Catmull-Rom spline is a special kind of spline built from the Hermite cubic curve. It is defined only by a sequence of control points. The gradients are derived from the control points in a form such that the curve can be differentiated at the control points.

Assume that we have a sequence of at least 2 real control points $p_1, \ldots, p_n$. We obtain p_0 by mirroring p_2 at p_1 and p_{n+1} by mirroring p_{n-1} at p_n. This

mirroring implies that the spline will have gradient zero at the first and last control point. The Catmull-Rom spline described by $p_1, \ldots, p_n$ is obtained by choosing

$$m_i = \frac{p_{i+1} - p_{i-1}}{2} \tag{8.47}$$

for $i = 1, \ldots, n$ and dividing the sequence of control points into $n-1$ sections, i.e. the first section is defined by p_1, m_1, p_2, m_2, the second section by p_2, m_2, p_3, m_3 etc. Each section defines a Hermite cubic curve on $[0,1]$, so we have to shift the sections into their respective place to obtain the complete curve. The final curve is then C^∞ (differentiable infinitely often, i.e. smooth) between the control points and C^1 (differentiable) at the control points. It has in general no continuous second derivative at the control points. This means the curve is continuous and has no bends.

Example 8.7 *The automaton shown in Figure 8.3 computes a Catmull-Rom spline for the control points* $(0,4,2,3)$. *The function it computes is shown in Figure 8.4. The spline's curve* $c(t)$ *interpolates the control points, i.e.* $c(0) = 0$, $c(1) = 4$, $c(2) = 2$, $c(3) = 3$. *The gradient at* 0 *and* 3 *is zero. The first derivative of the curve exists everywhere in* $[0,3]$.

Like in the case of Bézier curves, we can produce curves in higher dimensional spaces by choosing appropriate control point vectors. We can also build Catmull-Rom spline based patches for 3d modeling, as we did above for Bézier patches.

8.5 B-splines

Catmull-Rom splines are C^1 everywhere but in general not C^2. A single Bézier curve is a polynomial and thus C^∞, but lower order Bézier curves can only approximate very simple curves and higher order Bézier curves are not suitable for approximation in practice. If we split a curve into multiple pieces to represent it by a sequence of lower order Bézier curves, e.g. cubic Bézier curves, then it is usually simple to obtain a C^1 curve but cumbersome to construct a curve with C^2 or above continuity. The construction of an approximating or interpolating C^k curve for a given natural number k is simpler using B-spline curves, as k can be given as a parameter when constructing a B-spline curve such that the construction will automatically yield a C^k curve regardless of the choice of control points. We will only sketch some important properties of B-splines and show that B-spline curves can be computed by PWFA, as an exhaustive discussion of B-splines would go beyond the scope of this thesis. There are many textbooks discussing B-splines, as B-splines are very popular in computer graphics. An introduction can e.g. be found in [41], on which we have also based the notations and formulae used in this section.

We call a two way infinite sequence of real numbers $(t_i) = (t_i)_{i=-\infty}^{\infty}$ a knot sequence, if it is monotonically increasing. The k-th *B-spline* basis function, where B stands for basis, of degree n is a function parameterized by a knot sequence (t_i) that is piece-wise defined by polynomials. It can be written recursively as

$$N_k^0(t) = \begin{cases} 1 & \text{for } t_{k-1} \leq t < t_k \\ 0 & \text{otherwise} \end{cases} \tag{8.48}$$

$$N_k^n(t) = \frac{t - t_{k-1}}{t_{n+k-1} - t_k} N_k^{n-1}(t) + \frac{t_{n+k} - t}{t_{n+k} - t_k} N_{k+1}^{n-1}(t) \quad . \tag{8.49}$$

This recursive formulation is called the Mansfield, de Boor, Cox recursion (cf. [41]).

Given another two way infinite sequence of real vectors (or in fact elements of any K vector space such that $\mathbb{R} \subset K$) $(P_i) = (P_i)_{i=-\infty}^{\infty}$ where $P_j \in \mathbb{R}^d$ for each $j \in \mathbb{Z}$ and some $d \in \mathbb{N}, d > 0$, we can define a B-spline curve $c(t)$ of degree n as

$$c(t) = \sum_{k=-\infty}^{\infty} P_k N_k^n(t) \quad . \tag{8.50}$$

where (P_i) is called the control point sequence.

We call a knot sequence (t_i) uniform, if the equation

$$t_{i+1} - t_i = t_{i+2} - t_{i+1} \tag{8.51}$$

holds for each natural number i. A B-spline basis function based on a uniform knot vector is called a uniform B-spline basis function (or shorter uniform B-spline). If we fix some natural number n and a uniform knot vector (t_i), then the B-splines N_k^n based on (t_i) are shifted versions of each other. If a B-spline is not based on a uniform knot sequence, we call it a non-uniform B-spline. Let $t = t_j$ for some knot vector (t_i) and $j \in \mathbb{Z}$. Then we call the number of occurrences of t in (t_i) the multiplicity of t in (t_i) (or shorter the multiplicity of t).

Some important properties of B-splines are:

- *Finite Support*: It is apparent from the definition that each function N_k^n is only non-zero in the interval $[t_{k-1}, t_{k+n}]$, i.e. it has finite support. This implies that each B-spline curve based on a control point sequence that has only finitely many non-zero elements also has finite support.

- *Strong convex hull property*: Each point on a B-spline curve of degree n lies in the convex hull of no more than $n+1$ nearby control points. If we choose $n+1$ adjacent control points as the same vector v, then we force the curve to interpolate v.

- *Continuity*: Each B-spline curve of degree n based on a knot vector (t_i) is at least C^{n-r} at knots with multiplicity r (and C^{∞} at any point that is not a knot). This means we can diminish the continuity of a curve at some knot t_i by repeating t_i in the knot sequence.

- *Degree elevation*: If some curve $c(t)$ can be represented as a B-spline curve of degree n, then it can also be represented as a B-spline curve of degree $n+1$.

- *Linear independence*: For each n the set of functions $N_j^n, j \in \mathbb{Z}$ is linear independent.

Thus it is an easy task to generate a curve that interpolates a sequence of control points and is C^n for any chosen n using a B-spline curve.

For the construction of a PWFA from a B-spline curve $c(t)$ of degree n, we will assume that only finitely many members of the control point sequence (P_i) are non-zero, i.e. the curve has finite support. This assumption is common in

applications. If all elements of (P_i) are zero, then the curve is the zero curve on some finite or empty interval, which is clearly computable by a PWFA. Now assume that P_i has non-zero elements. Then we can shift both sequences (t_i) and (P_i) such that the first non-zero element of (P_i) is P_1. Now assume that P_1 is the first non-zero element of (P_i). Let q be the greatest index of a non-zero element of (P_i). Then we can write the curve as

$$c(t) = \sum_{j=1}^{q} P_j N_j^n(t) \tag{8.52}$$

which in non-zero only in the interval $[t_0, t_{n+q}]$. Let $(t_i')_1^r$ be the sequence obtained from (t_i) by reducing (t_i) to the subsequence relevant for the support of $c(t)$ and making each knot unique (i.e. such that (t_i') has only knots of multiplicity 1). We assume that (t_i') has at least two members, otherwise $c(t)$ is undefined everywhere and we can construct a PWFA computing the empty set to display $c(t)$. If we restrict $c(t)$ to the interval $[t_j', t_{j+1}')$ for $1 \leq j < r$, then we obtain a polynomial, which we denote by $p_j(t)$. Thus $c(t)$ can be realized as

$$\begin{aligned} c(t) &= \bigcup_{j=1}^{r-1} \{(t, p_j(t)) | t \in [t_j', t_{j+1}')\} \\ &= \bigcup_{j=1}^{r-1} \left\{ \left(\frac{t+t_j'}{t_{j+1}'-t_j'}, p_j\left(\frac{t+t_j'}{t_{j+1}'-t_j'} \right) \right) \middle| t \in [0,1) \right\} \quad . \end{aligned} \tag{8.53}$$

As a translated and dilated polynomial is still a polynomial, equation 8.53 describes a union of a set of polynomials, where each polynomial is defined on the interval $[0,1)$. This does not completely coincide with the definition of a PWFA computed set, as the right boundary 1 is missing. If, as it is the common case, the curve is continuous, then this problem only effectively appears at the right boundary of the interval $[t_{r-1}', t_r']$. As the problem however only appears as a limit, it is of no concern in applications. Thus we can compute any B-spline curve using a PWFA in practice.

Example 8.8 *Assume that we want to compute a linear B-spline, i.e.* $c(t) = N_1^1(t)$ *for the uniform knot vector given by* $t_0 = 0, t_1 = 1$*, which is a uniform B-spline. The function can then be written as*

$$N_1^1(t) = \begin{cases} t & \text{for } 0 \leq t < 1 \\ 2-t & \text{for } 1 \leq t < 2 \\ 0 & \text{otherwise.} \end{cases} \tag{8.54}$$

This can be done by computing

$$\begin{aligned} N_1^1(t) &= \{(t,t) | t \in [0,1)\} \cup \{(t, 2-t) | t \in [1,2)\} \\ &= \{(t,t) | t \in [0,1)\} \cup \{(t+1, 1-t) | t \in [0,1)\} \quad . \end{aligned} \tag{8.55}$$

The automaton computing this function and the plot of the function is shown in Figure 8.6.

In the same way one obtains the uniform quadratic B-spline $N_1^2(t)$ *as*

$$N_1^2(t) = \begin{cases} \frac{1}{2}x^2 & \text{for } 0 \leq t < 1 \\ -x^2 + 3x - \frac{3}{2} & \text{for } 1 \leq t < 2 \\ \frac{1}{2}x^2 - 3x + \frac{9}{2} & \text{for } 2 \leq t < 3 \\ 0 & \text{otherwise,} \end{cases} \tag{8.56}$$

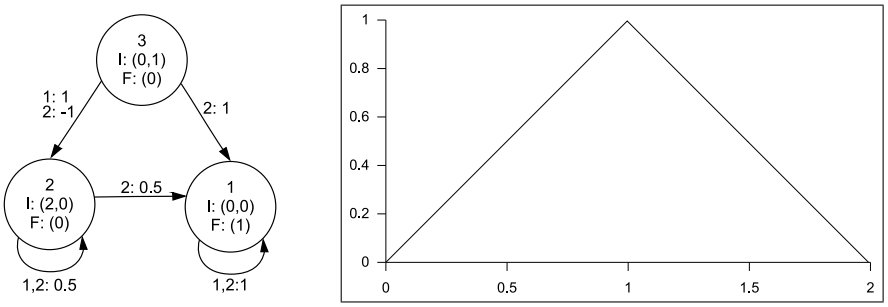


Figure 8.6: PWFA computing a uniform linear B-spline N_1^1 $(t_0 = 0, t_1 = 1, t_2 = 2)$ (left) and the function it computes on $[0,2]$ (right).

the uniform cubic B-spline $N_1^3(t)$ as

$$N_1^3(t) = \begin{cases} \frac{1}{6}x^3 & \text{for } 0 \le t < 1 \\ -\frac{1}{2}x^3 + 2x^2 - 2x + \frac{2}{3} & \text{for } 1 \le t < 2 \\ \frac{1}{2}x^3 - 4x^2 + 10x - \frac{22}{3} & \text{for } 2 \le t < 3 \\ -\frac{1}{6}x^3 + 2x^2 - 8x + \frac{32}{3} & \text{for } 3 \le t < 4 \\ 0 & \text{otherwise} \end{cases} \tag{8.57}$$

etc. An example of a non-uniform linear B-spline N_1^1 is obtained by choosing $t_0 = 0, t_1 = \frac{3}{2}$ and $t_2 = 2$. Then we obtain

$$N_1^1(t) = \begin{cases} \frac{2}{3}x & \text{for } 0 \le t < \frac{3}{2} \\ 4 - 2t & \text{for } \frac{3}{2} \le t < 2 \\ 0 & \text{otherwise.} \end{cases} \tag{8.58}$$

Another way to represent a B-spline curve using a PWFA is to first convert the curve to a sequence of Bézier curves and then construct an automaton for this sequence. A method for converting a B-spline curve to a set of Bézier curves was given by Böhm in [11].

The construction of B-spline patches of degree n given by

$$c(u,v) = \sum_{j=-\infty}^{\infty} \sum_{k=-\infty}^{\infty} P(j,k) N_j^n(u) N_k^n(v) \tag{8.59}$$

is similar to the curve case, even if the B-spline basis functions depending on u and v are not defined using the same knot vector.

8.6 Spline applications

Splines and spline patches have numerous applications ranging from computer fonts over computer aided design (CAD) as used in architecture, engineering, etc. to 3d modeling for computer generated animated movies. We will present some examples in this section.

8.6.1 Scalable Vector Graphics path outlines

We will show that the outline of a Scalable Vector Graphics (SVG, cf. [42]) path element can be displayed using a real PWFA of dimension 2, where the required graphical primitives can either be reproduced exactly or approximated at arbitrary precision.

An SVG path is defined using the following primitive instructions (we only provide informal descriptions of these instructions, the precise definitions can be found in [42]):

- M: move the pen to a new location and start a new path
- Z: close the current path by drawing a straight line from the current position to the starting point of the path
- L: draw a straight line from the current point to a given point
- H: draw a straight horizontal line from the current point up to a certain horizontal coordinate
- V: draw a straight vertical line from the current point up to a certain vertical coordinate
- C: draw a cubic Bézier curve from the current point to some endpoint, where two control points in between are given
- S: draw a cubic Bézier curve from the current point to some endpoint, where one control point in between is given and the other is computed from the previous curve segment such that the curve is C^1 at the current point
- Q: draw a quadratic Bézier curve from the current point to some endpoint, where one control point in between are given
- T: draw a quadratic Bézier curve from the current point to some endpoint, where the control point between the end points is computed from the previous curve segment such that the curve is C^1 at the current point
- A: draw an elliptical arc from the current point to some point

Each instruction exists in two variants. The first variant uses absolute coordinates. The instructions corresponding to this variant are denoted by upper case letters, e.g. the M command followed by a coordinate pair (x,y) moves the pen to the position (x,y) and starts a new path. The second variant uses relative coordinates (i.e. each coordinate pair denotes an offset relative to the last previously encountered coordinate pair). The instructions corresponding to this variant are denoted by lower case letters, e.g. the m command followed by a coordinate pair (x,y) moves the pen to the position (x_c+x, y_c+x) and starts a new path, if (x_c,y_c) is the current coordinate pair. The first instruction of a path has to be either an m or M, where an m is treated in the same way as an M (i.e. the current point assumed before the instruction is $(0,0)$).

An example of a path is:

M 0,0 C 0,-0.333 -0.333,-0.333 0,0

The path describes the cubic Bézier curve represented by the PWFA shown in Figure 4.2. The pen is first moved to the position $(0,0)$ and then a cubic Bézier curve is drawn to the endpoint $(0,0)$. The control points between the endpoints are $(0, -0.333)$ and $(-0.333, -0.333)$.

We will now reduce the number of possible instruction types by showing that each drawing instruction can be substituted either by a cubic Bézier curve or an elliptical arc, where only absolute coordinates are used. We use the following steps:

- Each instruction using relative coordinates can be converted into one that uses absolute coordinates and vice versa. We assume without loss of generality that each instruction uses absolute coordinates only, i.e. we consider paths that only use upper case letter instructions M,Z,H,V,L,C,S,Q,T and A.
- The Z, H and V instructions are only convenient notions of the L instruction that have some implicit parameters. Thus we can assume without loss of generality that each path we are provided with contains only the instructions M,L,C,S,Q,T and A.
- The spline variants S and T can be substituted by the more general variants C and Q respectively by explicitly inserting the implicit control point. This leaves the instructions M,L,C,Q and A.
- Bézier curves have the degree elevation property, i.e. each quadratic Bézier curve can be transformed into an equivalent cubic Bézier curve. More precisely, if a quadratic Bézier curve is given by the control points P_0, P_1 and P_2, then the cubic Bézier curve defined by the control points $P_0, (P_0 + 2P_1)/3, (2P_1 + P_2)/3, P_2$ defines the same curve. The remaining instruction types are M,L,C and A.
- A straight line going from P_0 to P_1 can be represented as a cubic Bézier curve using the control points $P_0, (P_0 + P_1)/2, (P_0 + P_1)/2, P_1$. Thus the instructions M,C and A remain.

The instruction set M,C,A is minimal in the sense that no instruction can be simulated exactly by combinations of the other two. We can however approximate the A instruction as a sequence of C instructions and vice versa at arbitrary precision. As the C instruction can be represented exactly using a PWFA, we assume that each encountered A instruction is replaced by a suitable sequence of C instructions and only the instructions M and C remain. In practice we may skip some of the steps above, e.g. representing a straight line by a cubic Bézier curve is unnecessarily complicated, as both can be computed by PWFA and the line can be implemented using a smaller amount of resources (states and edges) than the curve. As we have shown above, a PWFA can represent any finite set of Bézier curves. Thus each SVG path outline can be computed by a real PWFA of dimension 2.

8.6.2 Fonts

A typeface, which is also commonly called a font in computer based applications, mainly contains a function of non-empty words over a certain alphabet to

a set of shapes. The shapes are called glyphs. Usually each single alphabet letter is assigned a different glyph. Some fonts also contain glyphs describing words of a length greater than one, e.g. some fonts have a glyph for the word *fi*. Such glyphs for multi letter words are called ligatures. They are used, if the rendering of a word using two successive single glyphs does not look satisfactory. A font as a whole is in general endowed with some information describing the set of all contained glyphs. This information is called the font metrics. It contains values like baseline, ascent, descent, etc. (cf. [6]). Each glyph is defined as a relation on the Euclidean plane. When we set a line of text in a certain font, then we have to shift the rendered glyphs to their respective places on the line. This can be understood as a shift along the x axis for each rendered glyph. The shift between two adjacent glyphs a and b rendered consecutively in one line is given by the horizontal advance parameter of the glyph a. In some cases the sole usage of the horizontal advance parameter produces bad looking results, because two adjacent letters appear too close or too far apart. In this case the shift between pairs of glyphs can be changed by adding a so called kerning value.

The glyphs in most computer fonts are either stored as bitmaps or outlines. Bitmap fonts have low rendering complexity and can be optimized for a certain resolution, however they are in general not scalable. Outline fonts (which are sometimes also called vector fonts) store only glyph outlines, which have to be filled by the rendering algorithm. In contrast to bitmap fonts they in general are scalable, but the rendering requires much more complex algorithms. In particular, the rasterization of mathematically defined curves is a non-trivial task. An outline font may be obtained from a bitmap font by using a tracing algorithm. One such algorithm is the Potrace algorithm (cf. [82]). Another very simple tracing approach was given in [89]. The set of splines produced by such a tracing algorithm usually has to be improved by some amount of manual post processing.

Clearly, we can use PWFA for typesetting, if the single glyphs of a font can be represented using PWFA. Bitmap fonts can be rendered using ordinary WFA, as they are in fact nothing more than greyscale images. Thus we will put our emphasis on outline fonts, where we will use PWFA to represent glyph outlines as polynomial relations. We will not discuss the rasterization of such relations, but only the representation of the underlying curves in terms of PWFA.

There is a large variety of font file formats. The two most important formats in applications are the TrueType (cf. [74]) and Adobe PostScript type 1 (cf. [1]) font file formats. The TrueType format is the most common font format in Mac OS and Microsoft Windows. PostScript type 1 fonts, can be found in Adobe PostScript (cf. [2]) and PDF (Portable Document Format, cf. [3]) documents.

The outlines in a TrueType font are stored as quadratic B-splines. Thus we can either transform them directly to a PWFA or first transform the B-spline representation to a Bézier curve representation and then transform the Bézier curve representation into a PWFA. The drawing primitives allowed in the PostScript type 1 format are straight lines and cubic Bézier curves. Both can easily be transformed to PWFA form. Figure 8.7 shows a glyph outline consisting of cubic Bézier curves and straight lines. In practice the simplest way to transform a glyph outline of a TrueType or PostScript type 1 font into a PWFA is to use a font editor like FontForge (cf. [97]) to convert the font to an SVG font, which uses SVG paths to describe outlines, and then transform the respective path into a PWFA.

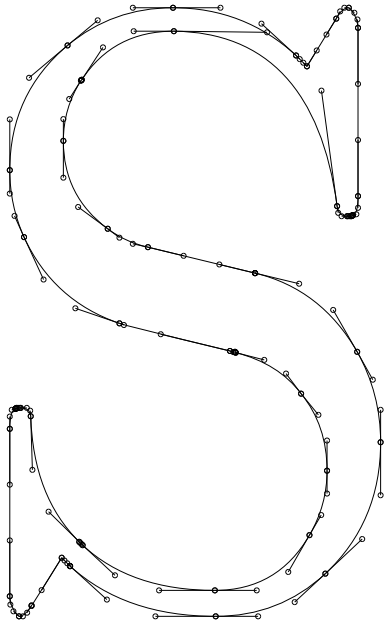


Figure 8.7: The outline of letter S of the Computer Modern font is displayed by cubic Bézier curves in the image. The control points are depicted as circles, the tangents are displayed as lines.

If we only represent the spline curves defining some glyph by a PWFA, then the description is incomplete, as the filling is missing. The following results show that PWFA can also be used to represent filled outlines defined by glyphs in PostScript type 1 or TrueType fonts.

Lemma 8.1 *Let $\mathcal{K} \supseteq \mathbb{R}$ be a field. Furthermore, let X be a PWFA of dimension d over $\mathcal{K}$ such that $S(X) = T(X)$. Then there exists a PWFA Y of dimension d over $\mathcal{K}$ computing the set*

$$S(Y) = \bigcup_{a,b \in S(X)} \{xa + (1-x)b | x \in [0,1]\}, \tag{8.60}$$

which is called the convex hull of $S(X)$.

Proof: Assume that $Q_X = \{1, \ldots, n\}$ for some $n \in \mathbb{N}^+$ and $\Sigma_X = \{1, \ldots, l\}$ for some $l \in \mathbb{N}^+$. Furthermore, assume without loss of generality that X is in final normal form and the single final state in X with final weight 1 is state n. Let U_k denote the identity matrix in $\mathcal{K}^{k \times k}$ for each positive natural number k. Let further $O_{k,m}$ denote the zero matrix in $\mathcal{K}^{k \times m}$ for each pair (k,m) of positive natural numbers. Consider the PWFA Y' of dimension $2d$ over $\mathcal{K}$ such that

- the state set $Q_{Y'}$ is given by $Q_{Y'} = \{1, \ldots, 2n\}$,
- the input alphabet $\Sigma_{Y'}$ is $\Sigma_{Y'} = \{1, \ldots, 2l\}$,

- the transition matrix $A_{Y'_i}$ is given by

$$A_{Y'_i} = \begin{cases} \begin{pmatrix} A_{X_i} & O_{n,n} \\ O_{n,n} & U_n \end{pmatrix} & \text{if } 1 \leq i \leq l \\ \begin{pmatrix} U_n & O_{n,n} \\ O_{n,n} & A_{X_{i-l}} \end{pmatrix} & \text{if } l+1 \leq i \leq 2l \end{cases} \tag{8.61}$$

  for $i = 1, \ldots, 2l$,

- the initial matrix $I_{Y'}$ is

$$I_{Y'} = \begin{pmatrix} I_X & O_{d,n} \\ O_{d,n} & I_X \end{pmatrix} \tag{8.62}$$

  and

- the final vector $F_{Y'}$ is $F_{Y'} = (F_X^T \; F_X^T)^T$.

Let $p(f_{Y'}(w))$ denote the projection of the vector $f_{Y'}(w)$ to the components $1, \ldots, l$ for each $w \in \Sigma_{Y'}^*$ and let $q(f_{Y'}(w))$ denote the projection of the vector $f_{Y'}(w)$ to the components $l+1, \ldots, 2l$ for each $w \in \Sigma_{Y'}^*$. Then apparently the set P given by

$$P = \overline{\bigcup_{w \in \Sigma_{Y'}^*} \{(p(f_{Y'}(w)), q(f_{Y'}(w)))\}} \tag{8.63}$$

is the set of all pairs (a,b) such that $a, b \in T(X)$. $S(Y)$ can be expressed as

$$S(Y) = \overline{\bigcup_{w \in \Sigma_{Y'}^*} \{p(f_{Y'}(w)) + x(q(f_{Y'}(w)) - p(f_{Y'}(w)))|x \in [0,1]\}} \; . \tag{8.64}$$

Thus Y can be obtained from Y' using the following steps:

1. Add two states $2n+1$ and $2n+2$ representing the real function $f(x) = x$ as described in Lemma 3.2. We assume that the state representing the linear function is state $2n+1$ and the state representing the constant function is $2n+2$.

2. Set the final vector to 1 for state $2n+2$ and to zero for each other state.

3. Let $c \in T(X)$ (such a point exists, because $S(X) = T(X)$ implies that $S(X)$ is not empty). Choose $I_Y = (I_X \; I_X \; O_{d,1} \; c)$ to halve the number of dimensions.

4. Insert a new label $2l+1$ and assign the transition matrix

$$A_{Y_{2l+2}} = \begin{pmatrix} U_n & O_{n,n} & -1_n & 1_n \\ O_{n,n} & U_n & 1_n & O_{n,1} \\ & & O_{2,2n+2} & \end{pmatrix} \tag{8.65}$$

   to it, where $1_n \in \mathbb{R}^n$ is 1 in component n and 0 otherwise.

Label $2l+1$ in Y is similar to the splitting label introduced in the constructed of a PWFA computing an iterated image as described in the proof of Theorem 5.11. Let $h(w) : \Sigma_Y^* \mapsto \Sigma_{Y'}^*$ denote the homomorphism that erases $2l+1$ and maps any other symbol to itself. When Y reads the symbol $2l+1$ after it has read

some word $w \in \Sigma_{Y'}^*$, then it configures the states computing the linear function to produce a line between the points $p(f_{Y'}(h(w)))$ and $q(f_{Y'}(h(w)))$. Thus Y is the set of all lines between pairs of (a,b) of points such that $a,b \in T(X)$. □

It is easy to see that Lemma 8.1 can be generalized to the following theorem.

Theorem 8.1 *Let X and Z be PWFA of dimension d over $\mathcal{K}$ such that $S(X) = T(X)$ and $S(Z) = T(Z)$, where $\mathcal{K} \supseteq \mathbb{R}$ is a field. Then there is a PWFA Y of dimension d over $\mathcal{K}$ computing the set*

$$S(Y) = \bigcup_{a \in S(X), b \in S(Z)} \{xa + (1-x)b | x \in [0,1]\} \ . \tag{8.66}$$

Proof: We do not construct the automaton Y' in the proof of Lemma 8.1 using two copies of X but one copy of X and one copy of Z. The rest of the construction is analogous. □

Using the result given in Theorem 8.1, we can construct PWFA that render filled glyphs defined by PostScript type 1 fonts. A TrueType font can usually be losslessly converted to a PostScript type 1 font. We assume without loss of generality that each used primitive is a cubic Bézier curve. Furthermore, we assume without loss of generality that there is exactly one connected area to be filled on the glyph. If there are multiple, we can handle them consecutively. The PostScript type 1 format does not allow outline paths to cross each other. This means that each area to be filled is defined by one path describing the outer border and a finite number of paths describing inner borders. We can approximate each glyph outline path from the inside of the glyph at arbitrary precision using a polygon. By the term from the inside we mean that each polygon vertex is on the glyph and no polygon edge crosses any glyph outline. We assume that no polygon approximating any outline path crosses itself or a polygon approximating any other outline path. If such a crossing exists, then this can be remedied by using more precise approximations. An example of such an approximation is shown in Figure 8.8. The depicted glyph has one outer and two inner borders. The area between the polygons approximating the outer border and inner borders can be tessellated using filled triangles. A filled triangle is the convex hull of any two edges of the triangle. This leaves the areas between the polygons and the glyph outlines. We have a closer look at how we obtain polygons approximating a glyph outline path from the inside. Observe the outline path shown in Figure 8.9. We want to approximate the curve C_1 by a polygon inside the area delimited by C_1, C_2 and C_3. Assume that C_1 is given as $C_1(t)$, where $C_1(0)$ denotes the lower end of C_1 and $C_1(1)$ the upper end in the figure. Let $N_1(t)$ denote the unit normal vector of C_1 at t, which points in the direction of the glyph inside. One such vector is depicted in the figure. We define the polygon $P_{1,k}(\varepsilon)$ approximating C_1 as the sequence of vectors

$$P_{1,k}(\varepsilon) = C_1(0), C_1\left(\frac{1}{k}\right) + \varepsilon N_1\left(\frac{1}{k}\right), \ldots, C_1\left(\frac{k-1}{k}\right) + \varepsilon N_1\left(\frac{k-1}{k}\right), C_1(1) \tag{8.67}$$

for $k \geq 2$ and $\varepsilon > 0$. If $P_{1,k}(\varepsilon)$ crosses C_1, then this can be remedied by increasing k, as C_1 is a polynomial and has only finitely many turning points. If we choose k sufficiently large and ε sufficiently small, then there are also no intersections between $P_{1,k}(\varepsilon)$ and C_2 or C_3. We can find a suitable pair of numbers (k,ε)

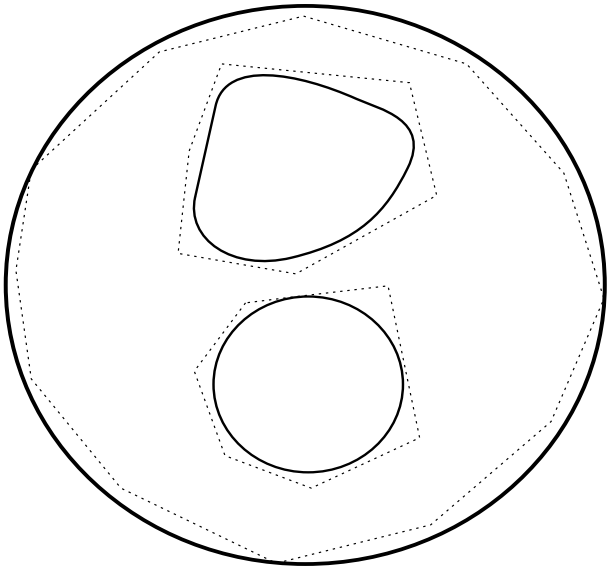


Figure 8.8: A glyph is approximated from the inside using polygons. The glyph outlines are displayed solid, the polygons as dashed lines.

algorithmically by e.g. trying each pair $(2^i, 2^{-i})$ for $i = 1, 2, \ldots$ until $P_{1,2^i}(2^{-i})$ no longer crosses C_1, C_2 and C_3. Note that only crossings are not allowed, the approximating polygon is allowed to touch each curve. Assume that we have found a suitable pair (k, ε). Then the sequence $P_{1,k}(\varepsilon)$ has length $k+1$. We have seen above that each Bézier curve $b(t)$ can be split into two curves $b_1(t)$ and $b_2(t)$ at any point $t \in (0,1)$ such that $b_1(t)$ and $b_2(t)$ together represent $b(t)$. Let $C_{1,1}, \ldots, C_{1,k}$ denote the sequence of cubic Bézier curves obtained by splitting C_1 at the points $\frac{1}{k}, \ldots, \frac{k-1}{k}$. Let further $L_{1,1}, \ldots, L_{1,k}$ denote the sequence of straight lines between the point pairs $(P_{1,k}(\varepsilon)_1, P_{1,k}(\varepsilon)_2), \ldots (P_{1,k}(\varepsilon)_k, P_{1,k}(\varepsilon)_{k+1})$. Then we can fill the area between the curve C_1 and the polygon $P_{1,k}(\varepsilon)$ by filling the areas between each pair $(C_{1,i}, L_{1,i})$ of delimiting curves using the construction given in the proof of Theorem 8.1.

8.6.3 3d modeling

Splines and spline patches are very popular in 3d modeling. We already showed that we are able to display textured spheres using PWFA. As we can display any type of textured spline surface using PWFA, we can also construct PWFA representing e.g. textured models of archaeological objects, as the following example shows.

Example 8.9 *We model an antique vase using textured spline patches. Assume that we are given the sequence of photographs as shown in Figure 8.10. We want to model the rotation symmetric part of the vase, thus we remove the background and vase handles from the images. As the handles in some pictures cover parts of the rotation symmetric body, we cannot remove them completely. The resulting image sequence is shown in Figure 8.11. Then we model the shape of the vase. As*

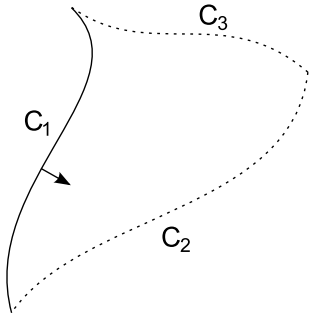


Figure 8.9: An outline path consisting of three cubic Bézier curves C_1, C_2 and C_3. C_1 is drawn through, C_2 and C_3 are dashed. The arrow depicts a normal vector of some point on the curve C_1 pointing into the glyph.

Figure 8.10: Photograph sequence of an antique Greek vase (*Amphora L 222*, courtesy of the *Martin von Wagner Museum der Universität Würzburg*) .

we consider the vase as a rotation symmetric object, the shape can be described by a single spline curve. Figure 8.12 shows a photo (with background and handles removed) of the vase next to a spline curve describing the vase. The spline curve was created manually from the photograph. The next step is the generation of spline patches from the spline curve. Just like we extended the display of a circle quadrant given in Example 8.3 to a spline patch displaying a sphere octant in Example 8.5, we can transform each single cubic Bézier curve used in the description of the vase outline to a set of four bicubic Bézier patches representing the corresponding part of the vase surface. Assume that a cubic Bézier curve to be transformed is given by

Figure 8.11: The background and parts of the handles that do not obscure the vase body have been removed from the images shown in Figure 8.10.

the control points P_0', P_1', P_2' and P_3' in the Euclidean plane. Let P_i for $i = 0,1,2,3$ denote the extension of P_i' to $\mathbb{R}^3$ by choosing the third component as 0. Let further $R \in \mathbb{R}^3$ denote the matrix representing a 90 degree rotation around the y axis. Then one of the four bicubic Bézier patches is given by the coefficient matrix

$$Q_0 = \begin{pmatrix} P_0 & P_0 + |P_0|(0\ 0\ k)^T & RP_0 + |P_0|(k\ 0\ 0)^T & RP_0 \\ P_1 & P_1 + |P_1|(0\ 0\ k)^T & RP_1 + |P_1|(k\ 0\ 0)^T & RP_1 \\ P_2 & P_2 + |P_2|(0\ 0\ k)^T & RP_2 + |P_2|(k\ 0\ 0)^T & RP_2 \\ P_3 & P_3 + |P_3|(0\ 0\ k)^T & RP_3 + |P_3|(k\ 0\ 0)^T & RP_3 \end{pmatrix}, \tag{8.68}$$

where k is chosen as in Example 8.3. The second matrix Q_1 is obtained from Q_0 by applying the matrix R to each single element of Q_0. Q_2 is obtained from Q_1 and Q_3 from Q_2 in the same way. The result of the transformation for the curve shown in Figure 8.12 is depicted in Figure 8.13.

A texture can be obtained by further processing the sequence of images shown in Figure 8.11. We only sketch the required steps. In practice there are various non-trivial problems like cylindric projections, image stitching etc. that have to be solved during the process. We assume that in each image the symmetry axis of the vase corresponds to a line which horizontally halves the image. The first step we take to obtain a texture from the images shown in Figure 8.11 consists of stretching the area covered by the vase in each picture to the complete image. This can be imagined as putting the vase into a sufficiently large cylinder such that the rotation axes of both objects match and then mapping each point on the vase to the closest point on the cylinder. The result of this step is shown in Figure 8.14. The projection produces heavy distortion on the left and right border of the images,

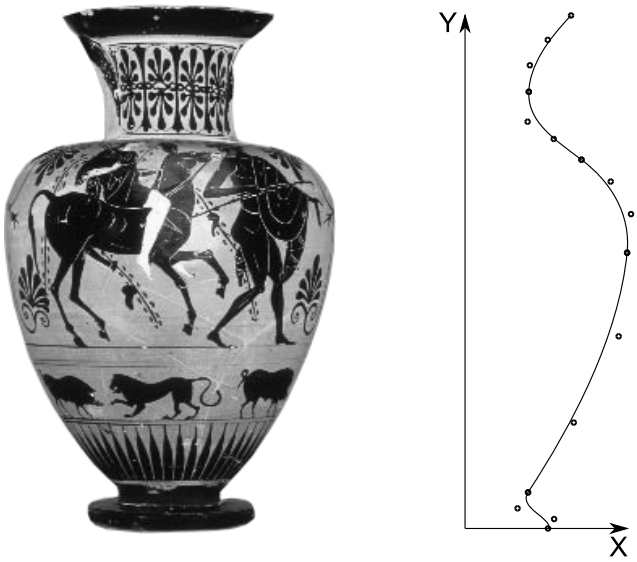


Figure 8.12: Photograph of an antique Greek vase after removing the background and the handles (left) and spline curve describing the vase built from 5 cubic Bézier curves (right). The spline only represents the rotation symmetric part of the vase body, i.e. the handles are missing. The control points of the spline curves are depicted as circles.

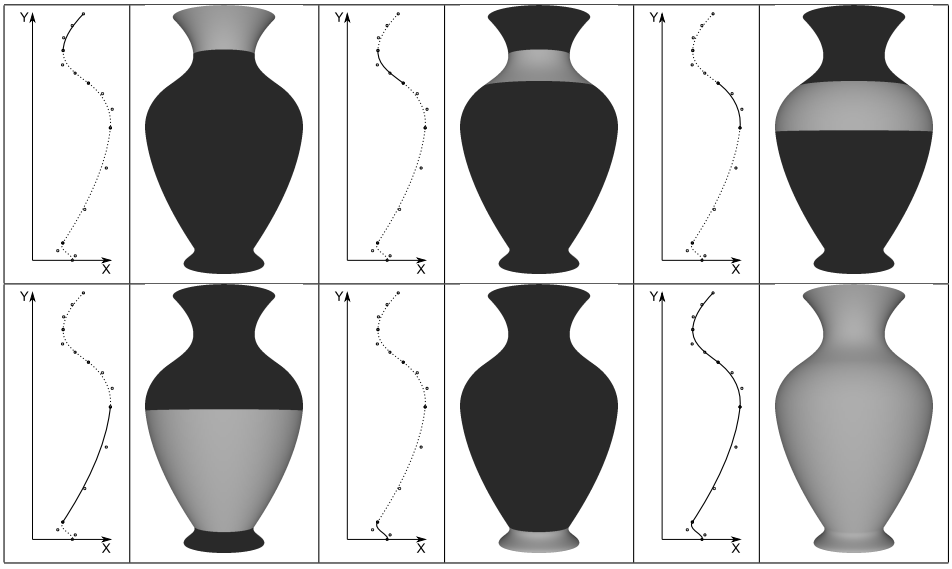


Figure 8.13: The spline curve depicted in Figure 8.12 is converted to a set of spline patches. Each cubic Bézier curve is converted to a set of 4 bicubic Bézier patches. The image pairs in the first and the first two image pairs in the second row each show one such conversion, where the translated parts are highlighted. The last image pair in the second row shows the complete conversion.

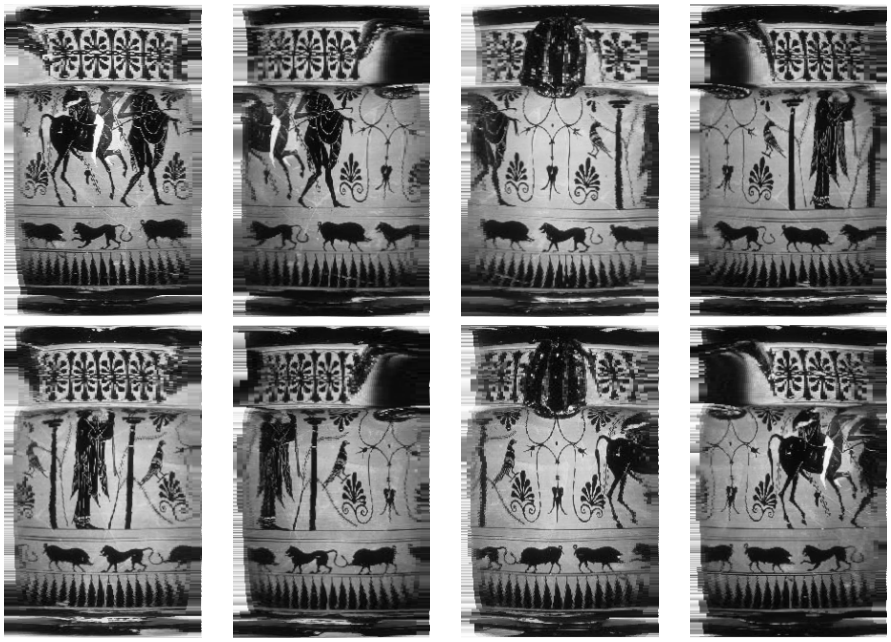


Figure 8.14: The images shown in Figure 8.11 have been projected onto a cylinder.

as the original images contain very little information on these areas. Therefor we remove these border areas from the images. The resulting images are depicted in Figure 8.15. In a next step the images are merged to a single texture. This process is usually called stitching*. Stitching algorithms are used e.g. in software generating panorama images from sets of photographs. The merged image shown in Figure 8.16 has been obtained by applying the Photomerge function of the Adobe Photoshop CS2 program to the images shown in Figure 8.15. The image in first column of the first row in Figure 8.15 was not used, because the remaining 7 pictures already show the complete outer surface of the vase. As the merged image shows some areas of the vase more than once, we need to clip it. The clipped image is shown in Figure 8.17. This clipped image is then partitioned into a set of subimages corresponding to the single bicubic Bézier patches describing the surface of the vase. This is depicted in Figure 8.18.*

We have constructed a set of textured bicubic Bézier patches. The union of these patches yields the complete description of the vase as a 3d model. Figure 8.19 shows a comparison of a vase photograph with a rendering of a 3d model of the vase displayed by a PWFA.

The page flip movies found on the DVD *Lorenz Fries - Chronik der Bischöfe von Würzburg* (cf. [44]) were also generated using spline models. An image of the rendered model is shown in Figure 8.20. The spline model employed for the construction of the book model was based on so called *Hash patches* (named after their inventor Martin Hash, cf. [57]). The movies were rendered with the Animation:Master program by Hash Incorporated. The texture images were obtained by taking photographs of a medieval manuscript (cf. [43]). As

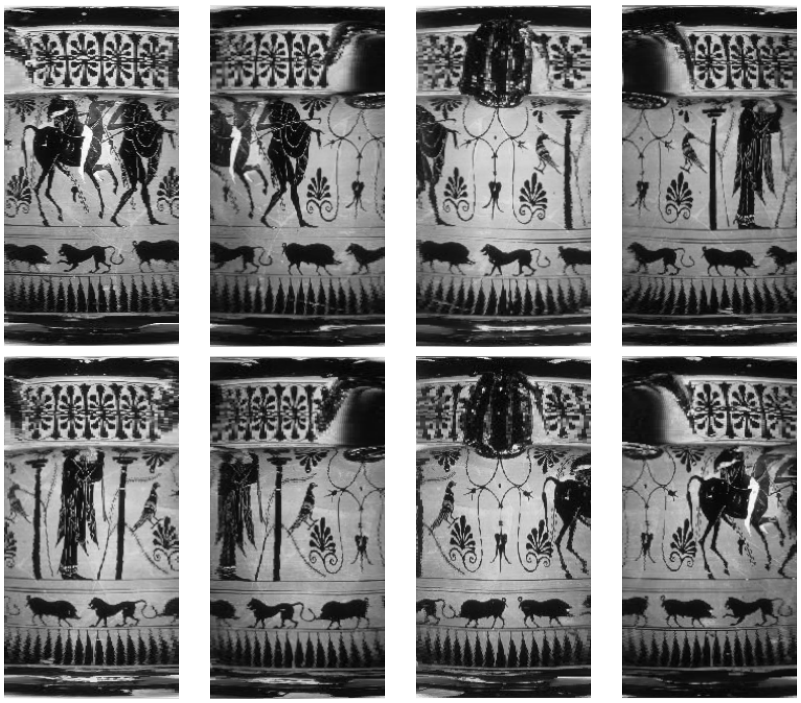


Figure 8.15: The images shown in Figure 8.15 have been clipped to remove the most heavily distorted areas at the left and right border.

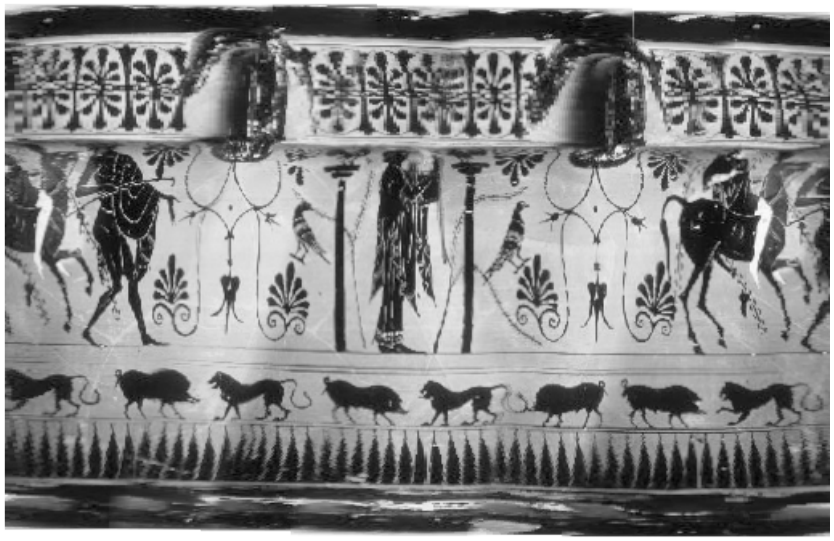


Figure 8.16: The images shown in Figure 8.15 have been used to compute a single merged image.

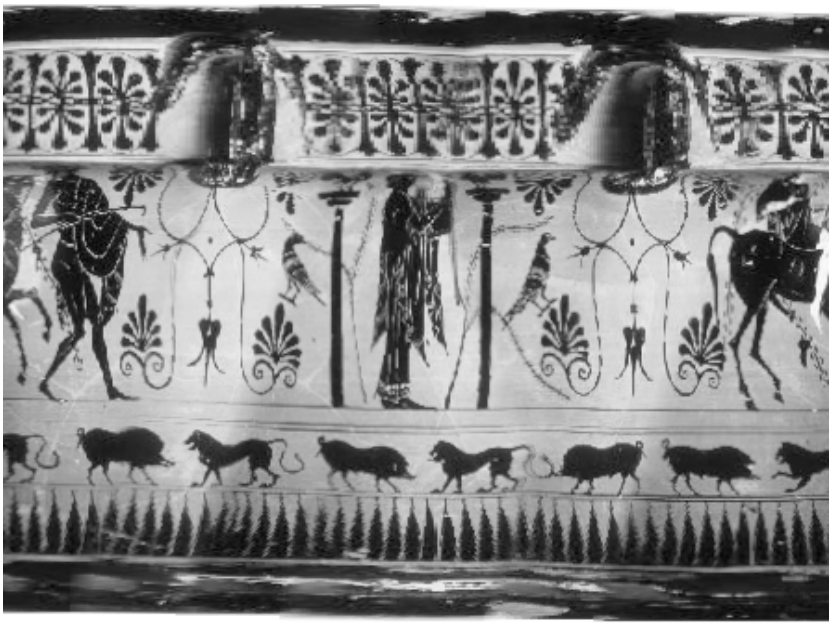


Figure 8.17: The image shown in Figure 8.16 has been clipped to display each part of the vase texture exactly once.

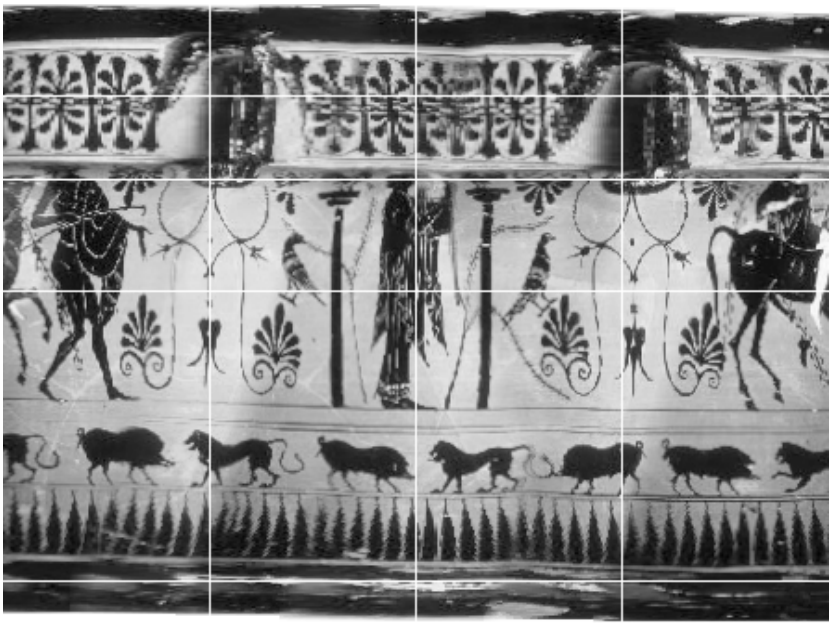


Figure 8.18: The complete texture shown in Figure 8.17 was split into the parts corresponding to the single bicubic Bézier patches.

Figure 8.19: Photograph of an antique Greek vase (left, *Amphora L 222*, courtesy of the *Martin von Wagner Museum der Universität Würzburg*) and rendering of a PWFA model of the vase (right).

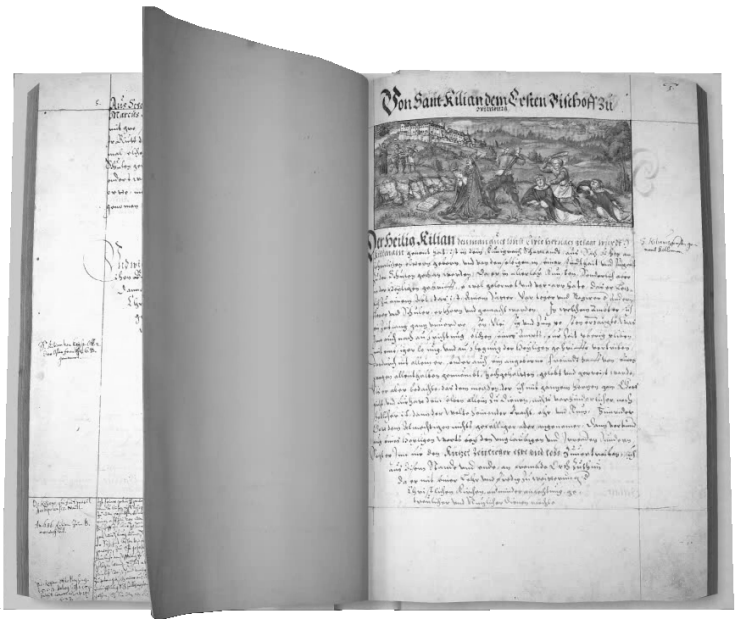


Figure 8.20: One frame of a page flip movie on the DVD [44], using scans of [43] as textures (courtesy of the *Universitätsbibliothek Würzburg*).

the rendering of the book model required high computational resources and thus a page flip could not be displayed in real time, each required page flip was rendered image per image and stored as a compressed movie. A WFA based video compression codec for compressing these movies was presented in [91].

Unlike Bézier patches, Hash patches are defined only by their boundaries. They are rendered in a subdivision process similar to the De Casteljau algorithm for Bézier curves, i.e. the patch is recursively divided until all obtained substructures sufficiently resemble polygons. There is no apparent correspondence between Hash patches and Bézier patches or B-spline patches, we can however approximate each Hash patch at arbitrary precision using a set of Bézier patches or B-spline patches and vice versa. Thus each book model can be represented by a PWFA at arbitrary precision.

Chapter 9

Interpretation and decoding

The set of vectors $S(X)$ computed by a PWFA X can be interpreted in a large variety of ways. We will discuss some possible interpretations in this chapter. In practice we can only evaluate the word function of a PWFA for a finite set of words. We have shown in subsection 5.2.1 that it is in general not decidable for a PWFA X, whether the set $S(X)$ or even $T(X)$ contains a certain vector v.

If however $S(X) = T(X)$ holds, then we know that every single vector computed by the word function is contained in $S(X)$. This means that in general the interpretation will be correct but possibly incomplete. Many automata used in applications have the property $S(X) = T(X)$. This ranges from automata computing polynomials, splines, spline patches and wavelets to automata obtained from WFA compressed pixel images.

A PWFA X constructed to represent a fractal (IFS, RIFS and MRFS) as described in Chapter 6) does in general not have the property $S(X) = T(X)$ but only the weaker property of ST-consistency. This means that for short words the vectors computed by the word function f_X may have a significant distance from the attractor of the represented fractal. In practice this effect can be attenuated by either not considering words that have a length below a given minimum or by substituting the final distribution F_X of the automaton by some vector $\mu_X(w)F_X$ for a suitable word $w \in \Sigma_X^*$ (or equivalently but computationally more expensive substitute the initial matrix I_X by $I_X\mu_X(w)$).

If a PWFA X does neither have the property $S(X) = T(X)$ nor is ST-consistent, then we can only compute and interpret a subset $\mathcal{T}$ of the set $T(X)$. In comparison to $S(X)$, this subset may be incomplete as well as incorrect. In contrast to ST-consistent automata, there are cases, where this cannot be improved or remedied by applying some sort of effectively computable preprocessing to the automaton, which is due to the undecidability results given in section 5.2.

9.1 Interpretation of PWFA computed sets

For the interpretation of a PWFA we will assume that every single vector v produced by a PWFA X, we interpret, is an element of $S(X)$. In the case of fractals (IFS,RIFS and MRFS) this may be only approximately true. As long as this effect however is smaller than e.g. the rounding errors we obtain by the use of finite

precision floating point arithmetic or some other form of inevitable quantization noise encountered in practice, it is negligible.

There are various orthogonal criteria by which we can separate the different proposed interpretations of vectors and sets. In some interpretations the input words are part of the interpretation, while in others it is unimportant which input word generated a certain vector. In the case of the classic WFA computed real functions the interpretation of the input word, which is performed using the function $r[k]$ for some integer k greater than 1, is also WFA computable. Thus, we can make the input word negligible by increasing the dimension of a PWFA. The same also applies to WFA computed functions of higher dimensions, e.g. compressed pixel images. Another way to separate the interpretations is by the range we map vectors to. Here the variants in practice include spaces like $\mathbb{R}^2$, $\mathbb{R}^2 \times \mathbb{R}, \mathbb{R}^2 \times \mathbb{R}^3, \mathbb{R}^3$ and $\mathbb{R}^{2^4}$. We will use point interpretation functions to map vectors produced by the word function of a PWFA to elements of some target space.

Definition 9.1 *Let X be a PWFA of dimension d over the semiring $\mathcal{S}$. Let $\mathcal{T}$ denote a set. Then we call each function $I : \mathcal{S}^d \mapsto \mathcal{T}$ a $\mathcal{T}$ (point) interpretation function for X.*

In many interpretations we augment vectors of spatial and temporal coordinates (we call each such vector a point) with vectors of scalars, e.g. we assign $(r\ g\ b)^T$ color vectors to points on the Euclidean plane. As PWFA compute relations, there is in general no reason to assume that each point is consistently assigned a unique scalar vector. Even if a PWFA assigns a unique scalar vector to every point it produces, this uniqueness does often no longer hold, after the single coordinates of the point have been quantized. This can be handled in several ways. The first way, which we call the overwriting strategy, is to choose exactly one set of scalars assigned to each point and discard all others. This is usually the last one produced by the used decoding algorithm, i.e. if a previously encountered point is assigned a new vector of scalars, the previous vector of scalars is overwritten with the new vector. A second way, which we call the averaging strategy, is to assign the average of all vectors of scalars produced for some point to the point. This can be performed using *averaging buffers*. An averaging buffer stores two things for each point. The first is the sum of all scalar vectors produced for the point. The second is the number of vectors assigned to the point. If we divide the summed up vectors by the number of vectors at the end of the decoding process, we obtain the average of all assigned scalar vectors for the point. Example 9.4 shows the difference between the two strategies by means of showing images that were obtained by interpreting the set computed by a single automaton with and without the usage of an averaging buffer.

9.1.1 Interpretation of sets as images

In this subsection an *image* or *picture* is, in contrast to the two dimensional words called pictures in [12], [47] and [73], some function from some cross product $[\alpha, \beta] \times [\gamma, \delta]$ of two real intervals $[\alpha, \beta], [\gamma, \delta]$ of positive diameter, i.e. a rectangle on the Euclidean plane to a set $[0, 1]^k$ for some positive natural number k.

Bi-level images

Let $\mathcal{A}$ denote a compact subset of the Euclidean plane, e.g. the attractor $\mathcal{A}$ of a 2d real fractal (IFS,RIFS or MRFS). Further let $B = [a,b] \times [c,d]$ denote a minimal axes-parallel bounding box for $\mathcal{A}$, i.e. a minimal axes-parallel rectangle on the Euclidean plane containing $\mathcal{A}$. Then we may identify the set $\mathcal{A}$ with the image $I : [a,b] \times [c,d] :\mapsto [0,1]$ defined by

$$I(i,j) = \begin{cases} 1 & \text{if } (i,j) \in \mathcal{A} \\ 0 & \text{otherwise.} \end{cases} \tag{9.1}$$

We call this image I the *bi-level image representing* $\mathcal{A}$.

Each bounded set computed by a PWFA over the reals is closed and thus compact. If a real PWFA computes an unbounded set, we can extract finite sets by clipping (i.e. by intersecting the set with a compact set). Such a finite set obtained by clipping again is compact. Thus we will for the interpretation of PWFA computed sets for the rest of the section without loss of generality assume that each PWFA computed set is compact.

The set $S(X)$ computed by a PWFA X of dimension 2 over $\mathbb{R}$ can be interpreted as a bi-level image, where the point interpretation function $I : \mathbb{R}^2 \mapsto \mathbb{R}^2$ for X is the identity.

Example 9.1 *The Koch (snowflake) curve is shown in Figure 9.1. The minimal axes-parallel bounding box of the curve is*

$$B = [0,1] \times \left[0, \frac{\sin(\frac{\pi}{3})}{3}\right]. \tag{9.2}$$

It is well known that the curve is continuous but nowhere differentiable.

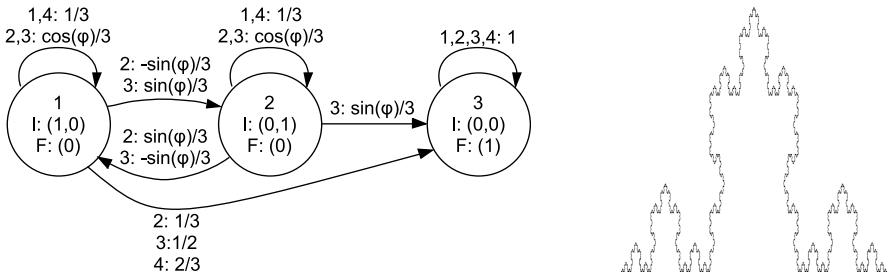


Figure 9.1: Automaton computing the Koch curve, where $\varphi = \frac{\pi}{3}$ (left) and its inverted interpretation (i.e. 0 is white, 1 is black) as a bi-level image (right).

In practice an approximation of the bounding box of a PWFA computed compact set can be obtained by computing the vectors produced by the corresponding PWFA up to a certain word length and computing the minima α, γ and maxima β, δ of the components of the coordinates. We denote this by the term *automatic clipping* or shorter auto clipping.

Greyscale images

A bi-level image can be extended to a greyscale image, where each point contained in a compact set is assigned a single real value that we interpret as the luminance of the point. Each point outside the compact set is assigned either the value 0 or 1 (i.e. the background of the image is set to black or white). The point interpretation function $I_{\text{Greyscale}} : \mathbb{R}^3 \mapsto \mathbb{R}^2 \times \mathbb{R}$ is given by

$$I_{\text{Greyscale}} \begin{pmatrix} x \\ y \\ l \end{pmatrix} = \begin{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \\ \begin{pmatrix} c(l) \end{pmatrix} \end{pmatrix} \tag{9.3}$$

for $x, y, l \in \mathbb{R}$ where $c : \mathbb{R} \mapsto [0,1]$ is defined as

$$c(l) = \begin{cases} 0 & \text{if } l < 0 \\ l & \text{if } 0 \leq l \leq 1 \\ 1 & \text{if } l > 1 \end{cases} \tag{9.4}$$

for $l \in \mathbb{R}$.

Remark 9.1 *The vector spaces $\mathbb{R}^3$ and $\mathbb{R}^2 \times \mathbb{R}$ are trivially isomorphic. We use the (syntactical) variant $\mathbb{R}^2 \times \mathbb{R}$ to optically separate the coordinates of a point from the assigned real value. The grouping of the components on the right-hand side of equation 9.3 has the same purpose.*

Example 9.2 *The automaton in Figure 9.2 computes the function $f(x,y) = (x+y)/2$ on $[0,1]^2$. The first and second component of the result vectors are interpreted as the x and y coordinates respectively. The third component is interpreted as the luminance of the point. The minimal axes-parallel bounding box is $[0,1]^2$.*

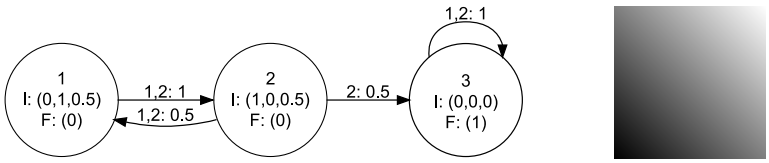


Figure 9.2: Automaton computing the function $f(x,y) = (x+y)/2$ (left) and its interpretation as a greyscale image (right).

Color images

Similar to a greyscale image, a bi-level image can be extended to a color image. Each point contained in a compact set is assigned three real values that we interpret as either the red, green and blue values (RGB) of the point or as one luminance and two chrominance values (YCbCr) of the point. In the YCbCr interpretation the Cb and Cr components usually assume a range of values that is centered at 0, i.e. are taken from the interval $[-0.5, 0.5]$. We shift these intervals by 0.5 to obtain values in $[0,1]$. Each point outside the compact set is assigned either the black or white color, i.e. the $(0\ 0\ 0)^T$ or $(1\ 1\ 1)^T$ triple in the

RGB case and the triple $(0\ 0.5\ 0.5)^T$ or $(1\ 0.5\ 0.5)^T$ in the (shifted) YCbCr case. The RGB and shifted YCbCr systems are linked by the relationships

$$Y = \alpha_R R + \alpha_G G + \alpha_B B \tag{9.5}$$

$$Cb = \frac{0.5}{1-\alpha_B}(B-Y) + 0.5 \tag{9.6}$$

$$Cr = \frac{0.5}{1-\alpha_R}(R-Y) + 0.5 \tag{9.7}$$

where $\alpha_R \approx 0.299$, $\alpha_G \approx 0.587$ and $\alpha_B \approx 0.114$, which can be written as the invertible affine transformation

$$\begin{pmatrix} Y \\ Cb \\ Cr \end{pmatrix} = \begin{pmatrix} 0.299 & 0.587 & 0.114 \\ -0.168736 & -0.331265 & 0.5 \\ 0.5 & -0.418688 & -0.081312 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix} + \begin{pmatrix} 0 \\ 0.5 \\ 0.5 \end{pmatrix} \tag{9.8}$$

(cf. [87]).

The point interpretation functions $I_{\mathrm{RGB}} : \mathbb{R}^5 \mapsto \mathbb{R}^2 \times \mathbb{R}^3$ and $I_{\mathrm{YCbCr}} : \mathbb{R}^5 \mapsto \mathbb{R}^2 \times \mathbb{R}^3$ are given by

$$I_{\mathrm{RGB}} \begin{pmatrix} x \\ y \\ R \\ G \\ B \end{pmatrix} = \begin{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \\ \begin{pmatrix} c(R) \\ c(G) \\ c(B) \end{pmatrix} \end{pmatrix} \tag{9.9}$$

for $x, y, R, G, B \in \mathbb{R}$ and

$$I_{\mathrm{YCbCr}} \begin{pmatrix} x \\ y \\ Y \\ Cb \\ Cr \end{pmatrix} = \begin{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \\ \begin{pmatrix} c(Y) \\ c(Cb) \\ c(Cr) \end{pmatrix} \end{pmatrix} \tag{9.10}$$

for $x, y, Y, Cr, Cb \in \mathbb{R}$, where the clipping function c is defined as above for greyscale images.

Example 9.3 *The automaton shown in Figure 9.3 computes a colored fractal dragon, i.e. each point on the attractor of the well-known dragon (cf. [8]) is attributed with 3 real values. Plates II and III in the appendix show the interpretation of the automaton as an RGB and YCbCr image respectively. The minimal axes-parallel bounding box is* $[0,1]^2$.

Example 9.4 *Let X denote the PWFA obtained by applying an iterated affine transformation representing a rotation by an angle of $\pi/4$ to the automaton X' shown in Figure 9.3. Then some points on the Euclidean plane are assigned multiple colors by X, if we interpret the automaton as an RGB image. Plate IV in the appendix shows the interpretation of X as an RGB image with and without the usage of an averaging buffer.*

The extension of RGB and YCbCr images to four channel RGBA and YCbCrA images by addition of an α value denoting the transparency or opacity of each point is straightforward.

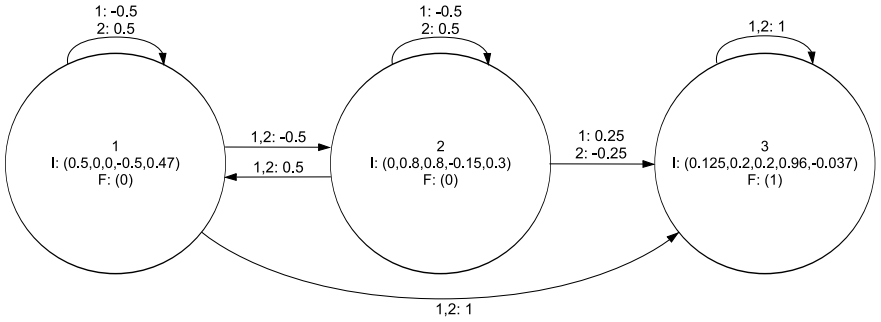


Figure 9.3: Automaton computing a colored fractal dragon.

9.1.2 Interpretation of sets as 3d scenes

Similar to a (2d) image we define a 3d scene (or 3d image) as a function

$$I : [\alpha,\beta] \times [\gamma,\delta] \times [\varepsilon,\zeta] \to [0,1]^k \tag{9.11}$$

where $\alpha,\beta,\gamma,\delta,\varepsilon$ and ζ are real numbers such that $\alpha < \beta, \gamma < \delta, \varepsilon < \zeta$ and k is a positive natural number. In analogy to the 2d case, we can define 3d bi-level, greyscale and color images. The corresponding point interpretation functions are

$$I_{\text{Bilevel3}}\left(\begin{pmatrix} x \\ y \\ z \end{pmatrix}\right) = \begin{pmatrix} x \\ y \\ z \end{pmatrix}, \tag{9.12}$$

$$I_{\text{Greyscale3}}\left(\begin{pmatrix} x \\ y \\ z \\ l \end{pmatrix}\right) = \begin{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} \\ \begin{pmatrix} c(l) \end{pmatrix} \end{pmatrix}, \tag{9.13}$$

$$I_{\text{RGB3}}\left(\begin{pmatrix} x \\ y \\ z \\ R \\ G \\ B \end{pmatrix}\right) = \begin{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} \\ \begin{pmatrix} c(R) \\ c(G) \\ c(B) \end{pmatrix} \end{pmatrix} \tag{9.14}$$

and

$$I_{\text{YCbCr3}}\left(\begin{pmatrix} x \\ y \\ z \\ Y \\ Cb \\ Cr \end{pmatrix}\right) = \begin{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} \\ \begin{pmatrix} c(Y) \\ c(Cb) \\ c(Cr) \end{pmatrix} \end{pmatrix} \tag{9.15}$$

where again the clipping function c is defined as above for greyscale 2d images.

We use a right handed orthogonal coordinate system to describe points $(x\ y\ z)^T$ in $\mathbb{R}^3$, which is depicted in Figure 9.4. As in the 2d case the x-axis is the horizontal and the y-axis the vertical axis. The direction of the positive z-axis is given by the right hand rule.

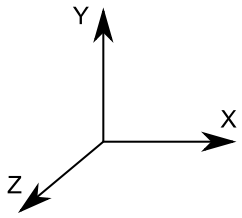


Figure 9.4: Right handed coordinate system

The process of rendering a 3d scene as a 2d image is called volume rendering. There are various volume rendering approaches, which can be divided into two categories:

1. Direct volume rendering: An algorithm given a 3d scene, a viewpoint and a line of sight directly computes a 2d image. The splatting and raycasting algorithms both fall into this category (cf. [96]).

2. Indirect volume rendering: An algorithm given a 3d scene transforms the scene into a different kind of 3d representation, which is subsequently rendered to a 2d image. This intermediate representation can e.g. be a set of spline patches or polygons, which describe surfaces in the 3d scene. The points in a 3d scene can be categorized, e.g. the points in a bi-level 3d scene can be divided into such points that are assigned the value 1 and such that are assigned the value 0. The borders between points falling into different categories can be interpreted as surfaces. A well known example of an algorithm transforming a (spatially quantized) bi-level 3d scene into a set of triangles is the marching cubes algorithm (cf. [68]). This process usually produces a very large number of triangles, which can only be rendered in real time using dedicated hardware.

A very simple approach to indirect volume rendering is the following: Let n_x, n_y, n_z be positive natural numbers and let $I : [\alpha,\beta] \times [\gamma,\delta] \times [\varepsilon,\zeta] \to [0,1]$ a bi-level 3d scene for $\beta > \alpha$, $\delta > \gamma$ and $\zeta > \varepsilon$. Then we can define the quantization functions $q_x : [\alpha,\beta] \mapsto \{1,\ldots,n_x\}$, $q_y : [\gamma,\delta] \mapsto \{1,\ldots,n_y\}$ and $q_z : [\varepsilon,\zeta] \mapsto \{1,\ldots,n_z\}$ as

$$q_x(x) = \left\lfloor \frac{(x-\alpha)(n_x-1)}{\beta-\alpha} + \frac{1}{2} \right\rfloor + 1 \tag{9.16}$$

for $x \in [\alpha,\beta]$,

$$q_y(y) = \left\lfloor \frac{(y-\gamma)(n_y-1)}{\delta-\gamma} + \frac{1}{2} \right\rfloor + 1 \tag{9.17}$$

for $y \in [\gamma,\delta]$ and

$$q_z(z) = \left\lfloor \frac{(z-\varepsilon)(n_z-1)}{\zeta-\varepsilon} + \frac{1}{2} \right\rfloor + 1 \tag{9.18}$$

for $z \in [\varepsilon,\gamma]$. Then we define the quantized 3d bi-level image $I_q : \mathbb{Z}^3 \mapsto [0,1]$ by

$$I_q(i,j,k) = \begin{cases} 1 & \text{if there exist } a \in [\alpha,\beta], b \in [\gamma,\delta], c \in [\varepsilon,\zeta] \text{ such that} \\ & q_x(a) = i, q_y(b) = j, q_z(c) = k \text{ and } I(a,b,c) = 1 \\ 0 & \text{otherwise.} \end{cases} \tag{9.19}$$

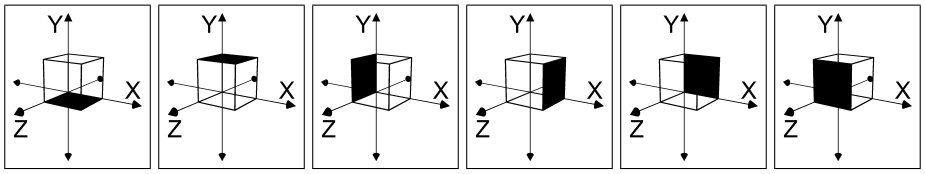


Figure 9.5: The rectangles bottom$(0,0,0)$, top$(0,0,0)$, left$(0,0,0)$, right$(0,0,0)$, back$(0,0,0)$ and front$(0,0,0)$ (left to right).

I_q is supported only on $D = \{1,\ldots,n_x\} \times \{1,\ldots,n_y\} \times \{1,\ldots,n_z\}$, we have defined it as 0 outside D for the sake of simpler notations in algorithms. For each $(i, j, k) \in \mathbb{Z}^3$ we define the quadrangles

$$\text{bottom}(i,j,k) = \left(\begin{pmatrix} i \\ j \\ k \end{pmatrix}, \begin{pmatrix} i+1 \\ j \\ k \end{pmatrix}, \begin{pmatrix} i+1 \\ j \\ k+1 \end{pmatrix}, \begin{pmatrix} i \\ j \\ k+1 \end{pmatrix} \right), \tag{9.20}$$

$$\text{top}(i,j,k) = \left(\begin{pmatrix} i \\ j+1 \\ k \end{pmatrix}, \begin{pmatrix} i+1 \\ j+1 \\ k \end{pmatrix}, \begin{pmatrix} i+1 \\ j+1 \\ k+1 \end{pmatrix}, \begin{pmatrix} i \\ j+1 \\ k+1 \end{pmatrix} \right), \tag{9.21}$$

$$\text{left}(i,j,k) = \left(\begin{pmatrix} i \\ j \\ k \end{pmatrix}, \begin{pmatrix} i \\ j+1 \\ k \end{pmatrix}, \begin{pmatrix} i \\ j+1 \\ k+1 \end{pmatrix}, \begin{pmatrix} i \\ j \\ k+1 \end{pmatrix} \right), \tag{9.22}$$

$$\text{right}(i,j,k) = \left(\begin{pmatrix} i+1 \\ j \\ k \end{pmatrix}, \begin{pmatrix} i+1 \\ j+1 \\ k \end{pmatrix}, \begin{pmatrix} i+1 \\ j+1 \\ k+1 \end{pmatrix}, \begin{pmatrix} i+1 \\ j \\ k+1 \end{pmatrix} \right), \tag{9.23}$$

$$\text{back}(i,j,k) = \left(\begin{pmatrix} i \\ j \\ k \end{pmatrix}, \begin{pmatrix} i+1 \\ j \\ k \end{pmatrix}, \begin{pmatrix} i+1 \\ j+1 \\ k \end{pmatrix}, \begin{pmatrix} i \\ j+1 \\ k \end{pmatrix} \right) \tag{9.24}$$

and

$$\text{front}(i,j,k) = \left(\begin{pmatrix} i \\ j \\ k+1 \end{pmatrix}, \begin{pmatrix} i+1 \\ j \\ k+1 \end{pmatrix}, \begin{pmatrix} i+1 \\ j+1 \\ k+1 \end{pmatrix}, \begin{pmatrix} i \\ j+1 \\ k+1 \end{pmatrix} \right). \tag{9.25}$$

These functions compute the sides of axes-parallel cubes of side-length 1 in $\mathbb{Z}^3$, e.g. front(i, j, k) is the front side of an axes-parallel cube of side-length 1 such that the vertex at the left backside of the bottom of the cube is located at position (i, j, k). Figure 9.5 shows graphical representations of the quadrangles bottom$(0,0,0)$, top$(0,0,0)$, left$(0,0,0)$, right$(0,0,0)$, back$(0,0,0)$ and front$(0,0,0)$. Using these definitions Algorithm 2 produces a set of quadrangles representing a quantized 3d image, where each position is assigned up to 6 quadrangles. The number of quadrangles produced by the algorithm is thus bounded by $6n_xn_yn_z$.

We can extend the construction of the quantized 3d scene I_q to the greyscale case. The extension to the color case is then obvious. Let $I : [\alpha,\beta] \times [\gamma,\delta] \times [\varepsilon,\zeta] \mapsto$

Input : Quantized bi-level 3d scene I_q with support
$\{1,\ldots,n_x\}\times\{1,\ldots,n_y\}\times\{1,\ldots,n_z\}$ for $n_x,n_y,n_z\in\mathbb{N}^+$.
Output: A set T of quadrangles representing I_q

```
1  T ← ∅
2  for z = 1 to n_z do
3      for y = 1 to n_y do
4          for x = 1 to n_x do
5              if I_q(x,y,z) ≠ I_q(x−1,y,z) then T ← T ∪ {left(x,y,z)}
6              if I_q(x,y,z) ≠ I_q(x+1,y,z) then T ← T ∪ {right(x,y,z)}
7              if I_q(x,y,z) ≠ I_q(x,y−1,z) then T ← T ∪ {bottom(x,y,z)}
8              if I_q(x,y,z) ≠ I_q(x,y+1,z) then T ← T ∪ {top(x,y,z)}
9              if I_q(x,y,z) ≠ I_q(x,y,z−1) then T ← T ∪ {back(x,y,z)}
10             if I_q(x,y,z) ≠ I_q(x,y,z+1) then T ← T ∪ {front(x,y,z)}
11 return T
```

Algorithm 2: MONO_QUANT(I_q,n_x,n_y,n_z): A quantized bi-level 3d scene I_q is transformed into a set of quadrangles.

$[0,1]$ be a greyscale 3d scene for $\alpha<\beta$, $\gamma<\delta$ and $\varepsilon<\zeta$. Let further $I':\mathbb{R}^3\mapsto[0,1]$ be defined as

$$I'(a,b,c)=\begin{cases} I(a,b,c) & \text{if } a\in[\alpha,\beta],b\in[\gamma,\delta],c\in[\varepsilon,\zeta] \\ 0 & \text{otherwise.}\end{cases} \tag{9.26}$$

Let n_x,n_y and n_z be positive natural numbers and the functions q_x, q_y and q_z be defined as above. Let further $q_x^{-1}(x)=\{i|q_x(x)=i\}$, $q_y^{-1}(y)=\{j|q_y(y)=j\}$ and $q_z^{-1}(z)=\{k|q_z(z)=k\}$. Then we define the quantized greyscale 3d scene $I_q:\mathbb{Z}^3\mapsto[0,1]$ as

$$I_q(i,j,k)=\sup\left\{g|g=I'(a,b,c) \text{ for } a\in q_x^{-1}(i),b\in q_y^{-1}(j),c\in q_z^{-1}(k)\right\}\quad . \tag{9.27}$$

Based on this definition of I_q, Algorithm 2 can be extended by assigning the greyscale value of a tuple $(i,j,k)\in\mathbb{Z}^3$ to all quadrangles produced for (i,j,k). Instead of using the supremum of I' over some interval in equation 9.27, we could also use the average value of I' inside the interval.

The following example presents some automata that can be interpreted as 3d scenes and some 2d images showing views of the scenes from certain viewpoints. The 2d images were generated by transforming quantized 3d scenes into sets of quadrangles using Algorithm 2, which were subsequently rendered using OpenGL. The shading algorithms of OpenGL were disabled to avoid the display of polygon edges. This delivers a better visualization of the original set by masking quantization effects.

Example 9.5 *The automaton X shown in Figure 9.6 computes a circle in the x, y plane of $\mathbb{R}^3$, i.e. the set $\{(x,y,0)|x^2+y^2=1,x,y\in\mathbb{R}\}$. Figure 9.7 shows the corresponding 3d scene rendered from two different viewpoints. Let $r:\mathbb{R}^3\mapsto\mathbb{R}^3$ be given by*

$$r(x)=\begin{pmatrix}1 & 0 & 0\\ 0 & \frac{4}{5} & \frac{3}{5}\\ 0 & -\frac{3}{5} & \frac{4}{5}\end{pmatrix}x \tag{9.28}$$

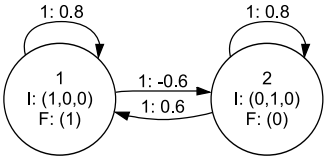


Figure 9.6: Automaton computing a circle in the x,y plane of $\mathbb{R}^3$.

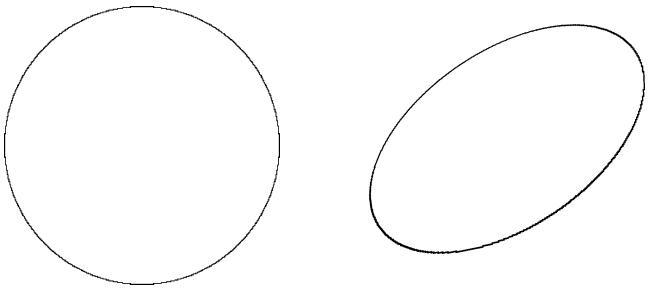


Figure 9.7: The (inverted, i.e. black is 1 and white is 0) scene computed by the automaton shown in Figure 9.6 viewed from a point on the positive z-axis right in front of the origin (left) and from a point $(x\ y\ z)^T \in \mathbb{R}^3$ such that $x > 0$, $y < 0$ and $z > 0$ (right).

for all $x \in \mathbb{R}^3$. Then the automaton X' shown in Figure 9.8 computes the iterated image of $S(X)$ under the single map r, which represents a rotation around the x-axis. The ratio of the angle $\arccos(\frac{4}{5})$ of the rotation and π is not a rational number. Thus X' computes the set $S(X') = \{(x\ y\ z)^T | x^2+y^2+z^2 = 1, x,y,z \in \mathbb{R}\}$, i.e. a sphere of radius 1 centered at the origin of $\mathbb{R}^3$. The interpretation of this set as a bi-level 3d scene is depicted on the left hand side of in Figure 9.9. X' is easily extended to compute greyscale or colored 3d scenes by adding 1 or 3 dimensions respectively. Rendered images obtained by this kind of extension are shown on the right-hand side of Figure 9.8 and in Plate V in the appendix for greyscale and RGB 3d scenes respectively.

The next example shows some images of rendered polygon sets, where the OpenGL shading algorithms were enabled. This shows the edges of the polygon model, i.e. the effects of the quantization that is necessary for the generation of the polygons are not masked.

Example 9.6 *We consider the Bézier patch presented in Example 8.5. Let I denote a bi-level 3d scene representing the patch. Let further $I_q(k)$ denote the quantized 3d scene of I for $n_x = n_y = n_z = k$ for each positive natural number k. Figure 9.10 shows images of polygon models obtained from $I_q(k)$ for various k using Algorithm 2 and the marching cubes algorithm.*

9.1.3 Interpretation of sets as polygonal surfaces

The interpretation functions above all map each domain vector to a single point in the range of the function. In most range spaces points are very small entities.

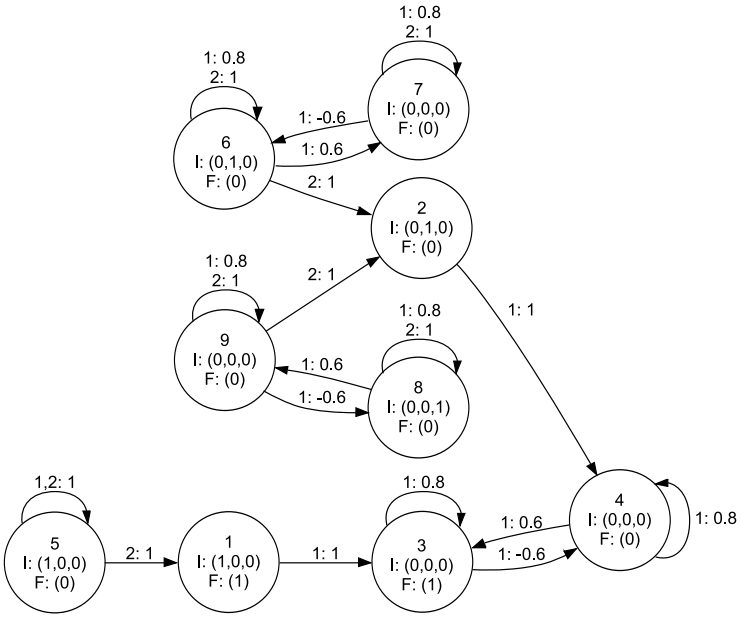


Figure 9.8: Automaton computing a sphere of radius 1 centered at the origin of $\mathbb{R}^3$.

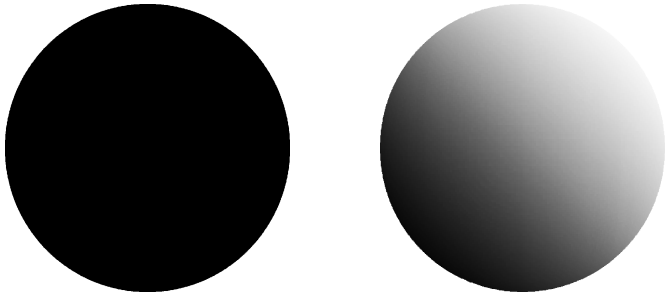


Figure 9.9: Left: The set computed by the automaton shown in Figure 9.8 is interpreted as an inverted bi-level scene and rendered from a point on the positive z-axis for $z > 1$ (i.e. outside the sphere) looking at the origin of $\mathbb{R}^3$. Right: The set computed by the automaton shown in Figure 9.8 is augmented by adding a fourth component to each vector $(x\ y\ z)^T$ that is assigned the value $\frac{x+y+2}{4}$. The image shows the interpretation of the augmented set as a greyscale 3d scene rendered from a point on the positive z-axis for $z > 1$ looking at the origin of $\mathbb{R}^3$.

These single points in practice are subsequently quantized for display or storage of images and 3d scenes. In the case of images the resulting entities are pixels and for 3d scenes we obtain voxels. The quality of this quantized representation depends heavily on the resolution of the employed quantizers, i.e. a 1024×1024 pixel image is in general able to show many more details of the original set than

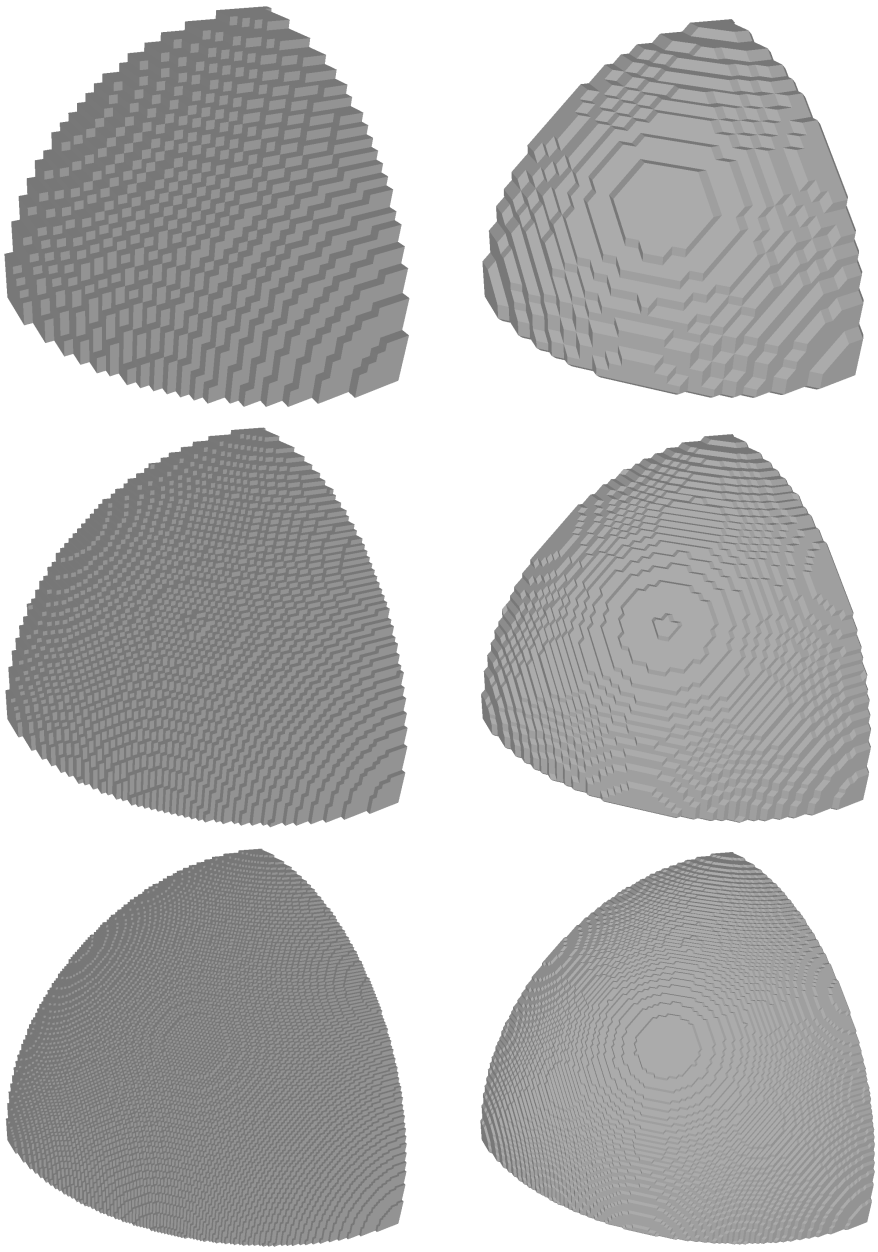


Figure 9.10: The Bézier patch presented in Example 8.5 rendered in the resolutions $n_x = n_y = n_z = 64$, 128 and 256 (top to bottom) using Algorithm 2 (left) and the marching cubes algorithm (right).

a 64×64 pixel image. The same applies to the quantization of greyscale and color values. In the case of uniform spatial quantization the amount of memory required for a pixel image grows quadratically with the resolution of the image. A 2048×2048 RGB uncompressed pixel image with 256 color levels per channel requires 12 megabytes of memory. At the time being 12 megabytes of memory are only a small fraction of the amount of random access memory found in contemporary personal computers, while a resolution of 2048×2048 pixels is sufficient to utilize the full capabilities of a computer display. In the case of a 3d scene however, the growth is cubic. This means that a scene with a resolution of 2048^3 color voxels with 256 color levels per channel has a memory consumption of 24 gigabytes, which cannot efficiently be handled by common contemporary hardware. While we can usually reduce this memory requirement by some tricks, e.g. representing connected areas that are assigned the same color as a whole and not storing each contained voxel, the memory requirement often still grows cubically with the resolution. A 3d scene is usually rendered to a 2d image for display by first applying an affine transformation and then projecting each point to its first two coordinates (cf. [96]). This maps a 3d scene consisting of n^3 voxels to an image consisting of roughly n^2 pixels, implying that we are unable to fully utilize the capabilities of a common computer display using a voxel model.

In 3d computer graphics objects are often represented by surfaces instead of volumes. There are various ways to describe surfaces. Two widely used tools for surface description are spline patches and polygons. We showed in section 8.3 that PWFA can be used to compute sets of quadrangles representing a Bézier patch. This can be generalized to the computation of arbitrary polygons. A polygon is given by a finite sequence of vertices $v_1, \ldots, v_k$, where pairs of adjacent vertices $(v_1, v_2), (v_2, v_3), \ldots, (v_k, v_1)$ are linked by edges. In our case each vertex is given as a vector of some real vector space of dimension $d \in \mathbb{N}, d > 0$ (usually $\mathbb{R}^3$). Such polygons can be computed by real PWFA of dimension dk, which are interpreted using the function $I_{\text{Bilevelpoly}} : \mathbb{R}^{dk} \mapsto (\mathbb{R}^d)^k$ defined as

$$I_{\text{Bilevelpoly}} \begin{pmatrix} v_{1,1} \\ \vdots \\ v_{1,d} \\ v_{2,1} \\ \vdots \\ v_{k,d} \end{pmatrix} = \begin{pmatrix} v_1 = \begin{pmatrix} v_{1,1} \\ \vdots \\ v_{1,d} \end{pmatrix} \\ \vdots \\ v_k = \begin{pmatrix} v_{k,1} \\ \vdots \\ v_{k,d} \end{pmatrix} \end{pmatrix}, \tag{9.29}$$

where the right-hand side of the equation is again divided into groups for better readability. We call $I_{\text{Bilevelpoly}}$ the bi-level polygonal interpretation function for PWFA. The greyscale, RGB and YCbCr polygonal interpretation functions are obtained by attributing each computed vertex with the appropriate number of scalars, i.e. the dimension of the automaton is increased. The greyscale and color components are again clipped to $[0,1]$ using the clipping function c as given in equation 9.4.

The representation of 3d objects as polygonal surfaces has some advantages and some drawbacks. The main advantage is that a polygonal surface of a voxel model can usually be rendered much faster than the volume itself. Although the

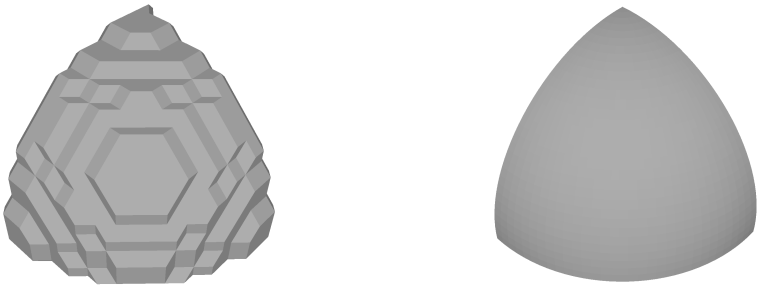


Figure 9.11: Both images display the Bézier patch presented in Example 8.5. The image on the left is obtained using the marching cubes algorithm for $n = n_x = n_y = n_z = 30$ to render a bi-level 3d scene representing the patch. It consists of 2176 triangles, of which roughly half are back-facing (i.e. invisible in the rendered image). The image on the right is rendered from a polygonal description of the patch as presented in Example 8.6 consisting of 1024 quadrangles.

marching cubes algorithm often represents a 3d scene by a very large number of triangles, the display of these triangles can be performed very quickly using dedicated hardware. The speed advantage is even bigger, if the 3d object is artificial and the surfaces were constructed as spline patches or polygons. The apparent drawback of representing a 3d object by a surface is that anything behind a surface is not displayed at all. This can be a negative point in e.g. medical applications, where some types of tissue are partly transparent.

Figure 9.11 shows that the polygonal display of a Bézier patch requires a much smaller number of geometric primitives for a satisfactory appearance than the display obtained from an indirect volume rendering of a 3d scene displaying the patch.

9.1.4 Interpretation of sets as movies

All of the above interpretations can be transformed into movies by adding one component representing a time index for each computed vector. Starting with a 2d image, we obtain a traditional movie consisting of a sequence of pictures, if we quantize the time component. The function $I_{\text{RGBMovie}} : \mathbb{R}^6 \mapsto \mathbb{R}^3 \times \mathbb{R}^3$ defined by

$$I_{\text{RGBMovie}} \begin{pmatrix} x \\ y \\ t \\ R \\ G \\ B \end{pmatrix} = \begin{pmatrix} \begin{pmatrix} x \\ y \\ t \end{pmatrix} \\ \begin{pmatrix} c(R) \\ c(G) \\ c(B) \end{pmatrix} \end{pmatrix} \tag{9.30}$$

for $x, y, f, R, G, B \in \mathbb{R}$ is an example of a movie interpretation function. In comparison to an RGB image, the only change is the additional t component, which describes a time index. This means that an RGB movie can be understood as an ordered set of RGB images.

Example 9.7 *Let X'' denote the automaton shown in Figure 9.3. If X'' is interpreted as an RGB image, then we obtain the image shown in Plate II in the appendix. Let P denote the real matrix given by*

$$P = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \tag{9.31}$$

and let $X' = \Xi[PI''_X](X'')$, i.e. the automaton obtained from X'' by inserting a null dimension between the second and third dimension. As X' has dimension 6, we can interpret it as an RGB movie. However X' produces only vectors for the time index 0. Let $R_\alpha(x)$ denote the affine transformation given by

$$R_\alpha(x) = \begin{pmatrix} \cos(\alpha) & \sin(\alpha) & 0 & 0 & 0 & 0 \\ -\sin(\alpha) & \cos(\alpha) & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} x + \begin{pmatrix} 0 \\ 0 \\ \alpha \\ 0 \\ 0 \\ 0 \end{pmatrix} \tag{9.32}$$

for each $\alpha \in \mathbb{R}$ and let $\beta = \operatorname{acos}(4/5)$. Further let X be any PWFA computing the iterated image of $S(X')$ under the two affine transformations R_β and R_{-1}. Then X represents a rotating colored fractal dragon. The third component of the vectors computed by X is the current rotation angle. If we interpret X as an RGB movie, then the time index equals the rotation angle. The interpretation of $S(X)$ clipped to the time interval $[0, 2\pi]$ as an RGB movie is visualized in Plate VI in the appendix.

9.2 Decoding

We present some PWFA decoding algorithms in this section. Each presented algorithm has its advantages and drawbacks. The algorithms differ in the amount of memory used and in their run-time. We will use the Landau symbol O (the so called big O notation) to give asymptotic upper bounds describing the memory and running time requirements of the algorithms (cf. [15]). The symbol O is often defined for functions of a single variable only, but the generalization to functions of multiple variables is straight-forward. The resources required by a certain algorithm to decode a PWFA often depend on multiple parameters (e.g. the number of states, the maximum length of input words to be processed, etc.). Thus the formulae we will state in the O notation to describe the complexity of the decoding algorithms will in general also be functions of multiple variables. In every case it is simple to restrict them to functions of a single variable by interpreting all other variables as constant parameters. As the membership problem for PWFA is undecidable, there is in general no parameter that could ensure that the set of vectors computed by a decoding algorithm is complete with regard to a desired output precision (e.g. such that the set of computed vectors provides a greyscale value for each pixel of a greyscale image of a given dimension). In some special cases however such a parameter exists. If we for example decode

a pixel image coded as a real WFA X over the alphabet $\{0,1,2,3\}$, then it is sufficient to evaluate the word function f_X of X for all words of length k to decode the image of dimension $2^k \times 2^k$ represented by X.

Each presented algorithm has the parameter $\underline{N}$, which denotes a minimal wordlength to be considered. The test for a minimal wordlength is useful, if we decode a PWFA generated for an IFS, and the usual start vector 0 has a significant distance from the attractor of the system. By choosing some finite $\underline{N} > 0$, we can often reduce the maximal distance of the delivered vectors from the set computed by the automaton.

In practice not every algorithm decodes every PWFA equally well. We will give a discussion of this issue at the end of the section.

We assume that the primitive semiring operations plus $(+)$ and times $(\cdot)$ can be performed in $O(1)$, i.e. require constant time. Furthermore, we assume that every single element of a semiring can be represented using a finite amount of memory. These assumptions in general do not hold, e.g. they do not apply for every semiring containing the natural numbers, if the numbers are stored in some form of variable length positional notation. In practice we however mostly use finite precision floating point numbers for computations and in this case addition and multiplication can be performed in constant time and every single number is represented using a fixed finite amount of memory.

Let $\Sigma = \{1, \ldots, l\}$ denote a finite alphabet. Then Σ^* can be represented as a complete tree of degree l with infinite depth. We say that this tree is spanned by Σ. The root of the tree corresponds to the empty word ε. Each node represents one word and if some node corresponds to the word $w \in \Sigma^*$, then the children of this node correspond to the words $w1$ to wl. Most presented algorithms can be understood as partial traversals of this tree, where some are like depth first and some like breadth first traversals.

The output of each decoding algorithm will always be some finite sequence (t_i). If some algorithm provided with a PWFA X and some additional algorithm specific parameters computes a sequence $(t_i)_{i=1}^k$, then each member of this sequence will be a member of the set $T(X)$. For every given algorithm there is (in theory) some way of making it run infinitely long by setting some specific parameter to ∞ or 0. Most of the algorithms then produce an infinite sequence $(t_i)_{i=1}^\infty$. If we can trigger that a certain algorithm produces an infinite sequence $(t_i)_{i=1}^\infty$ by setting certain parameters, then we say that the algorithm decodes a PWFA X completely (or the algorithm is complete for X), if $(t_i)_{i=1}^\infty$ has the property that

$$\overline{\bigcup_{i=1}^{\infty} \{t_i\}} = T(X) \ . \tag{9.33}$$

Some presented algorithms are complete for every PWFA, some are only complete for a subset of all PWFA and some are not complete for any PWFA, but show reasonable behavior when computing a finite sequence (t_i).

As the membership problem for PWFA is undecidable, there is no algorithm that computes the set $S(X)$ or an adequate approximation of $S(X)$, if it is given an arbitrary PWFA X.

We formulate each decoding algorithm without considering an interpretation function. This simplifies the theoretical discussion. In practice each decoding algorithm is supplied with an appropriate interpretation function and the interpretation of each vector produced by the word function of the considered

PWFA is rendered directly into some output data structure (e.g. a pixel is set in an image). Thus the construction of the sequence (t_i) is not performed in applications. For this reason we do not consider the amount of memory required to store the sequence (t_i), when we analyze the memory consumption of a PWFA decoding algorithm.

When we state the complexity (run-time or space) of a PWFA decoding algorithm in the following, then the letters n, l and d will as usual denote the number of states, the number of labels and the dimension of the input automaton respectively.

9.2.1 Stack based decoding

Stack based decoding of PWFA corresponds to a depth first traversal of a finite sub-tree of the infinite tree spanned by the alphabet of a given PWFA. We use the definitions of the stack operations *STACK-EMPTY*, *PUSH* and POP as given in Cormen et al. (cf. [15]). The stack based decoding algorithm for PWFA is given in Algorithm 3. Let $X = (Q = \{1,\ldots,n\}, \Sigma = \{1,\ldots,l\}, W = (A_1,\ldots,A_l), I, F)$ denote a PWFA of dimension d over the semiring $\mathcal{S}$. Each element on the stack is a pair of a vector $v \in \mathcal{S}^n$ and an integer r, where the vector v was obtained as $v = \mu_X(w)F$ for some word $w \in \Sigma^r$.

Input : PWFA $X = (Q, \Sigma = \{1,\ldots,l\}, W = (A_1,\ldots,A_l), I, F)$, minimum wordlength $\underline{N}$, maximum wordlength $\overline{N}$ where $\underline{N} \leq \overline{N}$
Output: Sequence $(t_i)_{i=1}^k$ where $k = l^{\underline{N}} + \ldots + l^{\overline{N}}$ such that $\bigcup_{i=1}^k \{t_i\} = \{v | v = f_X(w)$ for some $w \in \Sigma^*$ such that $\underline{N} \leq |w| \leq \overline{N}\}$

```
1  k ← 0
2  top[S] ← 0
3  PUSH(S,(F,0))
4  while STACK-EMPTY(S) = FALSE do
5      (v,r) ← POP(S)
6      if r ≥ N then
7          k ← k+1
8          t_k ← Iv
9      if r+1 ≤ N̄ then
10         for i = l,...,1 do
11             PUSH(S,(A_i v, r+1))
12 return (t_i)_{i=1}^k
```

Algorithm 3: STACK_DECODE(X,$\underline{N}$,$\overline{N}$): Stack based decoding algorithm for PWFA.

If X has a unary alphabet, then we can omit the stack, which is shown in Algorithm 4.

In comparison to other PWFA decoding algorithms, the stack based variants have very low memory requirements. The stack employed in Algorithm 3 needs to keep no more than $\overline{N}$ elements at any time and is the only kind of storage used by the algorithm. Thus the memory requirement of the algorithm is in $O(n\overline{N})$. If we consider n as fixed, then we obtain $O(\overline{N})$.

Algorithm 3 corresponds to the depth first traversal of a complete tree of

Input : PWFA $X = (Q, \Sigma = \{1\}, W = (A_1), I, F)$, minimum wordlength $\underline{N}$, maximum wordlength $\overline{N}$ where $\underline{N} \leq \overline{N}$

Output: Sequence $(t_i)_{i=1}^{\overline{N}-\underline{N}+1}$ such that $t_i = f_X(1^{i+\underline{N}-1})$ for $i = 1, \ldots, \overline{N}-\underline{N}+1$

```
1  j ← 0
2  k ← 0
3  v ← F
4  while k ≤ N̄ do
5      if k ≥ N then
6          j ← j + 1
7          t_j ← Iv
8      v ← A_1 v
9      k ← k + 1
10 return (t_i)_{i=1}^{j}
```

Algorithm 4: STACK_DECODE_UNARY(X,$\underline{N}$,$\overline{N}$): Stack based decoding algorithm for PWFA with unary alphabet.

degree l with depth $\overline{N}$, i.e. each node that is not a leaf has l children and a node is a leaf if and only if it has a distance of $\overline{N}$ from the root of the tree. The body of the while loop starting at line 4 is executed

$$l^0 + l^1 + l^2 + \ldots + l^{\overline{N}} = \frac{l^{\overline{N}+1} - 1}{l - 1} \tag{9.34}$$

times, once for each node of the tree. Assuming $\underline{N} = 0$, the matrix/vector product Iv is computed in every iteration, where each single product can be computed using $O(dn)$ semiring additions and multiplications. Each node of the tree is pushed onto the stack once. This happens in the lines 3 and 11 of the algorithm, where line 3 is executed once and line 11

$$l^1 + l^2 + \ldots + l^{\overline{N}} = \frac{l^{\overline{N}+1} - 1}{l - 1} - 1 \tag{9.35}$$

times. The product $A_i v$ in line 11 can be computed using $O(n^2)$ semiring additions and multiplications. Thus the equation

$$R(n, l, \overline{N}) \in O\left(\frac{l^{\overline{N}+1} - 1}{l - 1}(dn + n^2) - n^2\right) \tag{9.36}$$

holds for the running time R of the algorithm, which is high compared to other strategies.

Let X be a PWFA with alphabet $\Sigma = \{1, \ldots, l\}$ for $l \geq 2$. Then the stack based decoding algorithm potentially does not decode X completely, if we set $\underline{N} = 0$ and $\overline{N} = \infty$, as the depth first traversal only considers words in 1^*.

9.2.2 Queue based decoding

The queue based decoding algorithm is very similar to the stack based algorithm. The only difference is that the stack is substituted by a queue. We

use the definitions of the queue operations *QUEUE_EMPTY*, *ENQUEUE* and *DEQUEUE* as given by Cormen et al. (cf. [15]). In contrast to [15], we assume that we are operating on an array of infinite length, i.e. length$[Q] = \infty$. Making this presumption, overflow cannot occur. In practice a similar behavior can be implemented using a dynamically growing array.

The complete algorithm is shown in Algorithm 5.

Input : PWFA $X = (Q, \Sigma = \{1, \ldots, l\}, W = (A_1, \ldots, A_l), I, F)$, minimum wordlength $\underline{N}$, maximum wordlength $\overline{N}$ where $\underline{N} \leq \overline{N}$
Output: Sequence $(t_i)_{i=1}^{k}$ where $k = l^{\underline{N}} + \ldots + l^{\overline{N}}$ such that $\bigcup_{i=1}^{k} \{t_i\} = \{v | v = f_X(w)$ for some $w \in \Sigma^*$ such that $\underline{N} \leq |w| \leq \overline{N}\}$

```
1  k ← 0
2  head[Q] ← 1
3  tail[Q] ← 1
4  ENQUEUE(S,(F,0))
5  while QUEUE-EMPTY(S) = FALSE do
6      (v,r) ← DEQUEUE(S)
7      if r ≥ N̲ then
8          k ← k+1
9          t_k ← Iv
10     if r+1 ≤ N̄ then
11         for i = 1,...,l do
12             ENQUEUE(S,(A_i v, r+1))
13 return (t_i)_{i=1}^{k}
```

Algorithm 5: QUEUE_DECODE(X,$\underline{N}$,$\overline{N}$): Queue based decoding algorithm for PWFA.

The variant for PWFA with unary alphabet is exactly the same as in the case of the stack based algorithm for unary PWFA.

The queue based decoding algorithm has a much larger memory requirement than the stack based algorithm. The maximal number of queued elements is in $O(l^{\overline{N}})$, i.e. it is exponential in $\overline{N}$. In comparison, the stack based variant uses only an amount of memory that is linear in $\overline{N}$ for the stack. The running time of both algorithms is in the same order of growth.

In contrast to the stack based algorithm, the queue based algorithm completely decodes every given PWFA $X = (Q, \Sigma, W, I, F)$, if we set $\underline{N} = 0$ and $\overline{N} = \infty$, as every node of the tree spanned by Σ is visited.

9.2.3 Vector caching based decoding

The vector caching based approach is described in [21] as a decoding algorithm for real WFA representing images. Let $X = (Q, \Sigma = \{1, \ldots, l\}, W = (A_1, \ldots, A_l), I, F)$ denote a PWFA of dimension d over the semiring $\mathcal{S}$. We define the sequence of vector sequences $(F_k)_{k=0}^{\infty}$ inductively by the equations

$$((F_0)_i)_{i=1}^{1} = F \tag{9.37}$$

and

$$((F_{k+1})_i)_{i=1}^{l^{k+1}} = A_1 F_{k,1}, A_2 F_{k,1}, \ldots, A_l F_{k,1}, A_1 F_{k,2}, \ldots, A_l F_{k,l^k} \tag{9.38}$$

for $k \in \mathbb{N}$ (where we use the notation $F_{k,i}$ for F_{k_i}). The sequence F_k apparently has length l^k for each k. The computation of F_{k+1} can be performed using l^{k+1} matrix/vector products for each $k \in \mathbb{N}$, if F_k is given. Each matrix/vector product can be computed using $O(n^2)$ basic operations.

Thus Algorithm 7, which computes the sequence F_k, uses $O\left(n^2 \frac{l^{k+1}-1}{l-1}\right)$ operations, while requiring an amount of memory in $O\left(n \frac{l^{k+1}-1}{l-1}\right)$. It uses the subroutine FSEQ_STEP shown in Algorithm 6.

Input : PWFA $X = (Q, \Sigma = \{1, \ldots, l\}, W = (A_1, \ldots, A_l), I, F)$, natural number k, sequence (F_k) of X
Output: Sequence F_{k+1} of X

```
1 i ← 0
2 for j = 1, ..., l^k do
3     for m = 1, ..., l do
4         i ← i + 1
5         F_{k+1,i} ← A_m F_{k,j}
6 return F_{k+1,i}
```

Algorithm 6: FSEQ_STEP(X,k,F_k): Computation of the sequence F_{k+1} of a PWFA X for $k \in \mathbb{N}$, where the sequence F_k is given.

Input : PWFA $X = (Q, \Sigma = \{1, \ldots, l\}, W = (A_1, \ldots, A_l), I, F)$, wordlength $k \in \mathbb{N}$
Output: Sequence F_k of X

```
1 (F_{0,1}) ← F
2 for i=1,...,k do
3     F_i ← FSEQ_STEP(X, i − 1, (F_{i−1}))
4 return F_k
```

Algorithm 7: FSEQ(X,k): Computation of the set F_k of a PWFA X for $k \in \mathbb{N}$.

Similarly we define the sequence of matrix sequences $(I_k)_{k=0}^{\infty}$ inductively by the equations

$$((I_0)_i)_{i=1}^{1} = I \tag{9.39}$$

and

$$((I_{k+1})_i)_{i=1}^{l^{k+1}} = I_{k,1}A_1, I_{k,1}A_2, \ldots, I_{k,1}A_l, I_{k,2}A_1, \ldots, I_{k,l^k}A_l \tag{9.40}$$

for $k \in \mathbb{N}$. I_k can be computed using $O\left(dn^2 \frac{l^{k+1}-1}{l-1}\right)$ operations and an amount of memory in $O\left(dn \frac{l^{k+1}-1}{l-1}\right)$. The routines ISEQ_STEP and ISEQ are defined in analogy to FSEQ and FSEQ_STEP respectively.

We at first neglect the sequence I_k and formulate the PWFA decoding algorithm shown in Algorithm 8, which we call stack based decoding algorithm using suffix caching for PWFA.

The values of f and i at the end of the for loop starting in line 4 of Algorithm 8 are obtained by simultaneously solving the equations 9.41 and 9.42

Input : PWFA $X = (Q, \Sigma = \{1, \ldots, l\}, W = (A_1, \ldots, A_l), I, F)$, minimum wordlength $\underline{N}$, maximum wordlength $\overline{N}$ where $\underline{N} \leq \overline{N}$

Output: Sequence $(t_i)_{i=1}^{k}$ where $k = l^{\underline{N}} + \ldots + l^{\overline{N}}$ such that $\bigcup_{i=1}^{k} \{t_i\} = \{v | v = f_X(w)$ for some $w \in \Sigma^*$ such that $\underline{N} \leq |w| \leq \overline{N}\}$

$f \leftarrow 0$
$i \leftarrow 0$
$F_{0,1} \leftarrow F$
for $N = 1, \ldots, \overline{N}$ **do**
    if $l^{f+1} \leq d l^{i+1}$ **then**
        $f \leftarrow f + 1$
        $F_f \leftarrow$ FSEQ_STEP$(X, f-1, F_{f-1})$
    else
        $i \leftarrow i + 1$
$m \leftarrow 0$
$top[S] \leftarrow 0$
PUSH$(S, (I, 0))$
while STACK-EMPTY(S) = FALSE **do**
    $(J, k) \leftarrow$ POP(S)
    if $k + 1 \leq i$ **then**
        for $j = l, \ldots, 1$ **do**
            PUSH$(S, (JA_j, k+1))$
    for $j = 0, \ldots, f$ **do**
        if $j + k \geq \underline{N}$ **then**
            for $p = 1, \ldots, l^j$ **do**
                $m \leftarrow m + 1$
                $t_m \leftarrow JF_{j,p}$
return $(t_j)_{j=1}^{m}$

Algorithm 8: SUFFIX_CACHED_STACK_DECODE$(X, \underline{N}, \overline{N})$: Stack based decoding algorithm using suffix caching for PWFA.

and subsequently rounding f up and i down.

$$\frac{l^{f+1}-1}{l-1} = d\frac{l^{i+1}-1}{l-1} \tag{9.41}$$

$$\overline{N} = i+f \tag{9.42}$$

This yields (using some negligible simplifications)

$$f = \min\left\{\left\lceil\frac{\overline{N}+\log_l d}{2}\right\rceil, \overline{N}\right\} \quad . \tag{9.43}$$

The value of i is then obtained as $i = \overline{N} - f$. This choice of f and i minimizes the number of matrix/vector products computed in the algorithm (where we consider the multiplication of a $d \times n$ matrix with an $n \times n$ matrix as a sequence of d matrix/vector products in this argumentation).

We consider the memory requirements and the run-time of Algorithm 8.

In comparison to the pure stack based algorithm, the caching based algorithm uses a vector cache F_k for some k, which requires a large amount of memory. The total amount of required memory is in $O\left(n\frac{l^{f+1}-1}{l-1} + ni\right)$. This means that the memory requirements of the algorithm are exponential in f, which in typical applications is approximately $\overline{N}/2$, implying a total growth in the order of $O\left(n\sqrt{l^{\overline{N}}}\right)$.

The run-time $R(n,l,d,\overline{N})$ is given by the equation

$$R(n,l,d,\overline{N}) \in O\left(n^2\frac{l^{f+1}-1}{l-1} + dn^2\frac{l^{i+1}-1}{l-1} + dn\frac{l^{\overline{N}+1}-1}{l-1}\right). \tag{9.44}$$

Assuming that $f = i = \overline{N}/2$, we obtain

$$R(n,l,d,\overline{N}) \in O\left(n^2\frac{\sqrt{l^{\overline{N}}}-1}{l-1}(1+d) + dn\frac{l^{\overline{N}+1}-1}{l-1}\right). \tag{9.45}$$

In practice it turns out that the first summand is mostly negligible in comparison to the second. In comparison to the pure stack based algorithm, the remaining second summand in the running time of the algorithm is linear and not quadratic in the number of states.

Just like in the case of the pure stack based decoding algorithm, the stack based decoding algorithm using suffix caching does not completely decode all PWFA. In the special formulation of Algorithm 8, the for loop starting in line 4 is never left, if we set $\overline{N} = \infty$. The algorithm thus never adds any vectors to the sequence (t_i).

The queue based algorithm using suffix caching shown in Algorithm 9 does not suffer from this problem. It has the same running time as the stack based algorithm using suffix caching. Similar to the uncached case, the queue based variant has larger memory requirements than the stack based variant. When the algorithm processes the for loop starting at line 11, then the queue contains only such matrices that appear as members of the sequences I_i and I_{i+1} at any time. Thus it effectively uses the sequence of matrix sequences $(I_k)_{k=0}^{\infty}$.

Input : PWFA $X = (Q, \Sigma = \{1, \ldots, l\}, W = (A_1, \ldots, A_l), I, F)$, minimum wordlength $\underline{N}$, maximum wordlength $\overline{N}$ where $\underline{N} \leq \overline{N}$

Output: Sequence $(t_i)_{i=1}^{k}$ where $k = l^{\underline{N}} + \ldots + l^{\overline{N}}$ such that $\bigcup_{i=1}^{k} \{t_i\} = \{v | v = f_X(w)$ for some $w \in \Sigma^*$ such that $\underline{N} \leq |w| \leq \overline{N}\}$

$f \leftarrow 0$
$i \leftarrow 0$
$m \leftarrow 0$
$F_{0,1} \leftarrow F$
$head[Q] \leftarrow 1$
$tail[Q] \leftarrow 1$
ENQUEUE(Q,I)
if $\underline{N} = 0$ **then**
    $m \leftarrow m + 1$
    $t_m \leftarrow IF$
for $N = 1, \ldots, \overline{N}$ **do**
    if $l^{f+1} \leq dl^{i+1}$ **then**
        $f \leftarrow f + 1$
        $F_f \leftarrow$ FSEQ_STEP(X,$f-1$,F_{f-1})
    else
        for $j = 1, \ldots, l^i$ **do**
            $J \leftarrow$ DEQUEUE(Q)
            for $k = 1, \ldots, l$ **do**
                ENQUEUE(Q,JA_k)
        $i \leftarrow i + 1$
    if $\underline{N} \leq N$ **then**
        for $p = 1, \ldots, l^i$ **do**
            $J \leftarrow$ DEQUEUE(Q)
            ENQUEUE(Q,J)
            for $o = 1, \ldots, l^f$ **do**
                $m \leftarrow m + 1$
                $t_m \leftarrow JF_{f,o}$
return $(t_j)_{j=1}^{m}$

Algorithm 9: SUFFIX_CACHED_QUEUE_DECODE(X,$\underline{N}$,$\overline{N}$): Queue based decoding algorithm using suffix caching for PWFA.

In practice we have observed that the performance of the decoding algorithms using suffix caching described above is unnecessarily bad in some applications. Consider a PWFA of some dimension $d \in \mathbb{N}^+$ computing a set representing the union of a large number of polynomial curves of some degree q. Without loss of generality let this number be 2^k for some positive natural number k. The canonical construction of such an automaton using a binary alphabet produces $(q+1)+d(2^k-1)$ states, where $q+1$ states represent a polynomial of degree q and $d(2^k-1)$ states are used to build d binary trees of depth $k-1$. Then we observe that the vectors that are elements of the sequence F_{k+1} are very dense. This is in sharp contrast to the rows of the matrices that are elements of the sequence I_{k+1}, where each row is a vector that has at most $q+1$ non-zero components corresponding to the states of the automaton representing the polynomial of degree q. Thus the sequence I_{k+1} can be computed much more quickly then the sequence F_{k+1}, if specialized data structures and algorithms for sparse matrices and vectors are used. On the other hand a converse case, i.e. F_{k+1} can be computed much more quickly than I_{k+1}, can easily be constructed. If the dimension of the automaton is 1, then it is sufficient to transpose the transition matrices and to substitute the initial matrix by the transposed final distribution and vice versa. If such an imbalance exists in a PWFA, Algorithm 10 shows a much better run-time performance than Algorithm 8. The CLOCK function used in Algorithm 10 computes the number of primitive operations that have been executed between the beginning of the algorithm and the call of the CLOCK function. If the algorithm is implemented on a Turing machine, this equals the number of transitions the finite control of the machine has made up to the point where the CLOCK function is called. In practice we do not have to count the number of operations, but can use some kind of asynchronously running clock measuring the processing time of a program. The clock function of the C standard library is a suitable real world implementation of such functionality.

9.2.4 Chaos game based decoding

The chaos game algorithm can be used for rendering attractors of hyperbolic iterated function systems (IFS). It is described in [8]. As the automata in the class cPWFA represent linear transformations of attractors of affine IFS, it can also be used to decode certain PWFA. The chaos game algorithm is fundamentally different from the stack based, queue based and vector caching algorithms, because it is probabilistic.

Definition 9.2 *Let $p_1, \ldots, p_l$ be a finite sequence of non-negative real numbers such that $\sum_{i=1}^{l} p_i = 1$. Then we call $(p_1\ p_2 \ldots p_l)$ a probability vector (as usual l is called the dimension of the vector). We denote the set of all probability vectors of dimension l by $\mathcal{P}(l)$. If furthermore $p_i > 0$ for each $i = 1, \ldots, l$, then we say that $(p_1\ p_2 \ldots p_l)$ is irreducible and denote the set of all irreducible probability vectors of dimension l by $\mathcal{P}^+(l)$.*

Definition 9.3 *We call a sequence $(r_j)_{j=0}^{\infty}$ a random sequence generated by the probability vector $p = (p_1 \ldots p_l)$, if*

1. *$r_j \in \{1, \ldots, l\}$ for each $j \in \mathbb{N}$ and*
2. *each sequence member is determined by an independent statistical experiment that produces the value i with probability p_i for $i = 1, \ldots, l$.*

Input : PWFA $X = (Q, \Sigma = \{1, \ldots, l\}, W = (A_1, \ldots, A_l), I, F)$, minimum wordlength $\underline{N}$, maximum wordlength $\overline{N}$ where $\underline{N} \leq \overline{N}$

Output: Sequence $(t_i)_{i=1}^{k}$ where $k = l^{\underline{N}} + \ldots + l^{\overline{N}}$ such that $\bigcup_{i=1}^{k} \{t_i\} = \{v | v = f_X(w)$ for some $w \in \Sigma^*$ such that $\underline{N} \leq |w| \leq \overline{N}\}$

$m \leftarrow 0$
$f \leftarrow 0$
$i \leftarrow 0$
$fclock \leftarrow 0$
$iclock \leftarrow 0$
$F_{0,1} \leftarrow F$
$I_{0,1} \leftarrow I$
if $\underline{N} = 0$ **then**
    $m \leftarrow m + 1$
    $t_m \leftarrow IF$
while $f + i < \overline{N}$ **do**
    $tclock \leftarrow$CLOCK()
    if $fclock \leq iclock$ **then**
        // compute the final vector cache sequence F_{f+1}
        $f \leftarrow f + 1$
        $k \leftarrow 0$
        for $p = 1, \ldots, l^{f-1}$ **do**
            for j=1,...,l **do**
                $k \leftarrow k + 1$
                $F_{f,k} \leftarrow A_j F_{f-1,p}$
        $fclock \leftarrow$CLOCK()$-tclock$
    else
        // compute the initial matrix cache sequence I_{i+1}
        $i \leftarrow i + 1$
        $k \leftarrow 0$
        for $o = 1, \ldots, l^{i-1}$ **do**
            for j=1,...,l **do**
                $k \leftarrow k + 1$
                $I_{i,k} \leftarrow I_{i-1,o} A_j$
        $iclock \leftarrow$CLOCK()$-tclock$
    if $i + f \geq \underline{N}$ **then**
        for $o = 1, \ldots, l^{i}$ **do**
            for $p = 1, \ldots, l^{f}$ **do**
                $m \leftarrow m + 1$
                $t_m \leftarrow I_{i,o} F_{f,p}$
    $fclock \leftarrow \frac{fclock}{l}$
    $iclock \leftarrow \frac{iclock}{dl}$
return $(t_j)_{j=1}^{m}$

Algorithm 10: CACHE_DECODE(X,$\underline{N}$,$\overline{N}$): Decoding algorithm using prefix and suffix caching for PWFA.

We denote the set of all random sequences generated by a probability vector p by $\mathcal{R}(p)$.

The algorithm is presented in Algorithm 11. We assume that the parameter $\underline{N}$ is chosen as 0, for the rest of the subsection. This simplifies notations without fundamentally changing the provided results.

Input : PWFA $X = (Q, \Sigma = \{1, \ldots, l\}, W = (A_1, \ldots, A_l), I, F)$, minimum wordlength $\underline{N}$, maximum wordlength $\overline{N}$ such that $\underline{N} \leq \overline{N}$, random sequence $(r_j)_{j=0}^{\infty}$ generated by an irreducible probability vector of dimension l
Output: Some sequence $(t_i)_0^{\overline{N}-\underline{N}}$, where for each $i = 0, \ldots, \overline{N} - \underline{N}$ the equation $t_i = I\mu_X(w_i)F$ holds for at least one $w_i \in \Sigma^{\underline{N}+i}$

```
k ← 0
v_0 ← F
for i = 0, ..., N̄ do
    if i ≥ N then
        t_k ← I v_i
        k ← k + 1
    v_{i+1} ← A_{r_i} v_i
return (t_i)_{i=0}^{k-1}
```

Algorithm 11: CHAOS_GAME_DECODE(X,$\underline{N}$,$\overline{N}$,$(r_j)_{j=0}^{\infty}$): Chaos game based decoding algorithm for PWFA.

Definition 9.4 *Let $(\mathcal{S}^d, m)$ be a complete metric space for some $d \in \mathbb{N}^+$, where $\mathcal{S}$ is a semiring. Furthermore, let $X = (Q, \Sigma = \{1, \ldots, l\}, W, I, F)$ be a PWFA of dimension d over $\mathcal{S}$. Let (r_j) denote a random sequence generated by an irreducible probability vector $(p_1 \ldots p_l)$. Then we call the infinite sequence $(t_i)_{i=0}^{\infty}$ given by*

$$t_i = \begin{cases} f_X(\varepsilon) & \text{for } i = 0 \\ f_X(r_0 \ldots r_{i-1}) & \text{otherwise} \end{cases} \tag{9.46}$$

for each $i \in \mathbb{N}$ the infinite random walk of X under (r_j). The sequence $(t_i)_{i=0}^{N-1}$ is called the finite random walk of X under (r_j) with length N for each positive natural number N.

The chaos game based decoding algorithm apparently computes finite random walks. We may imagine that it computes infinite random walks, if we set $\overline{N}$ to infinity. Each such infinite random walk can be interpreted as a set as described in the following definition.

Definition 9.5 *Let $(\mathcal{S}^d, m)$ be a complete metric space for some $d \in \mathbb{N}^+$, where $\mathcal{S}$ is a semiring. Furthermore, let $X = (Q, \Sigma = \{1, \ldots, l\}, W, I, F)$ be a PWFA of dimension d over $\mathcal{S}$. Let further $(r_j) \in \mathcal{R}(p)$ for some irreducible probability vector p of length l. Let (t_i) the infinite random walk of X under (r_j). We say that (t_i) defines the set*

$$\mathcal{T}((t_i)) = \bigcap_{j=0}^{\infty} \overline{\bigcup_{i=j}^{\infty} \{t_i\}} \ . \tag{9.47}$$

Just like the set $S(X)$ computed by a PWFA X, the set $\mathcal{T}$ defined by an infinite random walk is given in a limes superior construction. However these two limits are defined using different sequences of sets. In the PWFA case we use the sets S_i, which have l^i elements. Each set $\{t_i\}$ used for the definition of a set defined by a random walk contains only a single vector.

Definition 9.6 *Let $(\mathcal{S}^d, m)$ be a complete metric space for some $d \in \mathbb{N}^+$, where $\mathcal{S}$ is a semiring. Furthermore, let $X = (Q, \Sigma = \{1, \ldots, l\}, W, I, F)$ be a PWFA of dimension d over $\mathcal{S}$.*

- *Let $p \in \mathcal{P}^+(l)$. We say that the chaos game based decoding algorithm is p sequence consistent for X, if there exists some set $\mathcal{T}_p(X)$, such that*

$$\mathcal{T}(\text{CHAOS\_GAME\_DECODE}(X, 0, \infty, (r_j))) = \mathcal{T}_p(X) \tag{9.48}$$

  holds with probability 1, if we randomly choose some $(r_j)_{j=0}^{\infty} \in \mathcal{R}(p)$, where each choice is equally likely.

- *We say that the chaos game based decoding algorithm is totally consistent for X, if it is p sequence consistent for every $p \in \mathcal{P}^+(l)$ and $\mathcal{T}_p$ is the same set for each $p \in \mathcal{P}^+(l)$. If total consistency holds, then we denote the unique computed set by $\mathcal{T}(X)$.*

Definition 9.7 *Let $(\mathcal{S}^d, m)$ be a complete metric space for some $d \in \mathbb{N}^+$, where $\mathcal{S}$ is a semiring. Furthermore, let X be a PWFA of dimension d over $\mathcal{S}$. We say that the chaos game based decoding algorithm for PWFA completely decodes X, if it is totally consistent for X and $\mathcal{T}(X) = S(X)$.*

Remark 9.2 *Definition 9.7 only requires that the chaos game based algorithm produces the set $S(X)$ defined by a PWFA X with probability 1. This is only an approximation of the condition formulated in equation 9.33.*

The following lemma states a well known result of IFS theory expressed in terms of PWFA.

Lemma 9.1 *Let $(\mathcal{S}^d, m)$ be a complete metric space for some $d \in \mathbb{N}^+$, where $\mathcal{S}$ is a semiring. Furthermore, let X be a PWFA of dimension d over $\mathcal{S}$, where X is based on contraction mappings. Then the chaos game based decoding algorithm decodes X completely.*

The chaos game based decoding algorithm is unsuitable for some PWFA, as the following example shows.

Example 9.8 *Let Y and Z be real PWFA of dimension 1 such that $\Sigma_Y = \Sigma_Z = \{1, 2\}$ and $S(Y) \cap S(Z) = \emptyset$, where Y and Z are based on contraction mappings. Let X be the real PWFA of dimension 1 computing $S(X) = S(Y) \cup S(Z)$ that is constructed as described in the proof of Theorem 5.1. Let s denote the single initial state of X. Then clearly the component s of the vector $I_X \mu_X(w)$ is null for each $w \in \Sigma_X^* \setminus \{\varepsilon\}$, i.e. state s only assumes a non-zero value for the empty word. Let X' be the PWFA obtained from X by choosing $I_{X'} = F_X^T$, $F_{X'} = I_X^T$ and $A_{X_i'} = A_{X_i}^T$ for $i \in \Sigma_X$. X' computes the same set as X, as $f_{X'}(w) = f_X(w^R)$, where w^R denotes the reversed word of w. Now consider the application of the chaos game based decoding algorithm to*

X'. After reading the first symbol, the algorithm will, regardless of the supplied random sequence, behave like decoding either the automaton Y or the automaton Z, but never like decoding both at the same time. Note however that the algorithm behaves correctly for the automaton X, although it is not based on contraction mappings. This means that membership of a PWFA in the class cPWFA is sufficient for decodability using the chaos game based algorithm, but it is not necessary.

The next lemma is a trivial implication of Example 9.8 and gives a necessary criterion for the decodability of a PWFA using the chaos game based algorithm

Definition 9.8 *Let $X = (Q = \{1, \ldots, n\}, \Sigma, W, I, F)$ be a PWFA of dimension d over the semiring $\mathcal{S}$ and let $F' \in \mathcal{S}^n$. Then $\Omega[F'](X)$ denotes the PWFA that is obtained from X by substituting the final distribution of X by F'.*

Lemma 9.2 *Let $(\mathcal{S}^d, m)$ be a complete metric space for some $d \in \mathbb{N}^+$, where $\mathcal{S}$ is a semiring. Furthermore, let $X = (Q, \Sigma, W, I, F)$ be a PWFA of dimension d over $\mathcal{S}$. If the equation*

$$S(X) = S(\Omega[\mu_X(w)F](X)) \tag{9.49}$$

does not hold for each $w \in \Sigma^*$, then the chaos game based decoding algorithm does not decode X completely.

Lemma 9.2 can be interpreted in the following way: when Algorithm 11 applies a randomly chosen transition matrix corresponding to some alphabet symbol $a \in \Sigma$ in line 7 during the i'th iteration of the for loop starting in line 3 and ignores all transition matrices corresponding to other possible alphabet symbols in $\Sigma \setminus \{a\}$, it does so based on the assumption that for every real number $\varepsilon > 0$, there is some $j > i$ such that $|Iv_j - Iv_i| < \varepsilon$. This means it relies on returning to an equivalent situation, whenever it makes a random choice, i.e. it assumes some kind of recurrence. If this recurrence does not always hold for a PWFA X, the algorithm does not decode X completely. Note that in contrast to the IFS case, the above inequality $|Iv_j - Iv_i| < \varepsilon$ is sufficient. We do not require the stronger constraint $|v_j - v_i| < \varepsilon$.

It is well known that as long as we are considering a PWFA in the class cPWFA and the chosen probability vector is irreducible, the concrete choice of the single probabilities p_i is unimportant, i.e. the chaos game algorithm decodes X completely. In practice a careful choice of each probability can greatly speed up the decoding process. Barnsley proposes a choice of

$$p_i \approx \frac{|\det A_i|}{\sum_{i=1}^{l} |\det A_i|} \tag{9.50}$$

in [8], if $\det A_i \neq 0$ and a small number like 0.001 otherwise, where det denotes the determinant. Another way of assigning the probability p_i is based on the eigenvalues of the matrix $A_{i,S} = A_i^H A_i$. Let $\lambda_{i,j}$ be the (not necessarily distinct) eigenvalues of $A_{i,S}$ for $j = 1, \ldots, n$. Then we first assign the value $\sum_{j=1}^{n} \lambda_{i,j}/l$ to the value p_i and then normalize the vector $(p_1 \ldots p_l)$ by dividing it by the sum of its components. As in Barnsley's construction, we have to make sure that the vector is irreducible by substituting a potential probability of zero with a small real number like 0.001. In both cases, we need to remove all states computing the constant function from the automaton, before we start the computation of

the probabilities p_i. If we neglect this step, then the probabilities are biased by the eigenvalue 1 introduced by the constant function.

We consider the space and run-time requirements of the algorithm. Let $(\mathcal{S}^d, m)$ be a complete metric space for some $d \in \mathbb{N}^+$, where $\mathcal{S}$ is a semiring. Furthermore, let $X = (Q = \{1, \ldots, n\}, \Sigma, W, I, F)$ be a PWFA of dimension d over $\mathcal{S}$. One of the main advantages of the chaos game based algorithm is that it has a very low memory demand in $O(n)$, as it requires only storage space for a finite number of vectors. The sequence (v_i) in Algorithm 11 does usually not exists in real implementations, we have solely introduced it to simplify the discussion of the algorithm. The statement

$$R(n, l, d, \overline{N}) \in O\left(\overline{N}(dn + n^2)\right) \tag{9.51}$$

holds for the run-time $R(n, l, d, \overline{N})$. It does not depend on the alphabet size l.

9.2.5 Metric based decoding

We have shown that the chaos game based algorithm fails to completely decode some PWFA in Example 9.8. On the other hand the stack, queue and caching based algorithms can in practice be applied for a small finite word length only and there are PWFA computing sets that are very inadequately described by any finite wordlength that can be handled by implementations of those algorithms on real computers.

The metric based decoding algorithm for PWFA can be understood as a modified version of the queue based algorithm. The queue based algorithm traverses a complete tree of depth $\overline{N}$ in breadth first order. This depth $\overline{N}$ is provided as a parameter to the algorithm. We can imagine this as a breadth first traversal of a complete tree of infinite depth, where we check at every node, whether the child nodes have a depth of at most $\overline{N}$. If the result of this test is negative, then we consider the node as a leaf, otherwise as an inner node. In the metric based algorithm this test is substituted by a check that determines, whether we have seen an equivalent node before during the traversal. This check is performed using a metric to compare a current vector with a set of previously seen vectors.

The metric based decoding algorithm is shown as Algorithm 12. It uses a queue to perform a modified breadth first tree traversal. The first vector inserted into the queue is the final distribution of the automaton, which is annotated with the word length 0. The algorithm produces at most $\mathcal{N}$ vectors. This is guaranteed by checking the number of produced vectors before each run of the while loop in line 6. When the algorithm has taken a vector v out of the queue in line 7, it first checks, whether this vector is contained in an ε ball of any vector v' it has previously handled and inserted into the set T. If such a vector v' is found, then v is discarded, as v and v' are assumed to be equivalent and we expect that the further treatment of v and its descendants obtained by multiplying v by any sequence of transition matrices does not produce anything new. If no such vector v' exists, we consider v as new. Then we queue its successors and if the vector was produced for a word of sufficient length, we add it to the set T and append the vector Iv to the output sequence (t_i).

If we set the parameter ε to 0, then Algorithm 12 degenerates to a (rather inefficient) variant of the queue based decoding algorithm. Thus the algorithm clearly decodes every given PWFA over each metric space $(\mathcal{S}, m)$ such that $\mathcal{S}$ is a semiring completely.

Input : PWFA $X = (Q, \Sigma = \{1, \ldots, l\}, W = (A_1, \ldots, A_l), I, F)$, positive real number ε, metric m, minimum word length $\underline{N}$, maximum number of generated vectors $\mathcal{N}$

Output: Sequence $(t_i)_{i=1}^{k}$ for some positive natural number k such that for every i the equation $t_i = Iv$ holds for some $v \in \mathcal{F}$, where $\mathcal{F}$ is a maximal set of vectors such that

- $|\mathcal{F}| \leq \mathcal{N}$,
- each $v \in \mathcal{F}$ is produced by f_X for at least one word $w \in \Sigma^{\underline{N}}\Sigma^*$ and
- $m(v_1, v_2) \geq \varepsilon$ holds for each pair of distinct vectors $(v_1, v_2) \in \mathcal{F}^2$.

```
1  k ← 0
2  T ← ∅
3  head[Q] ← 1
4  tail[Q] ← 1
5  ENQUEUE(Q,(F,0))
6  while (QUEUE-EMPTY(S) = FALSE) AND (|T| < N) do
7      (v,r) ←DEQUEUE(Q)
8      e ← ∞
9      foreach v' ∈ T do
10         e ← min{e,m(v,v')}
11     if e ≥ ε then
12         if r ≥ N then
13             T ← T ∪ {v}
14             k ← k+1
15             t_k ← Iv
16         for i = 1,...,l do
17             ENQUEUE(Q,(A_i v, r+1))
18 return (t_i)_{i=1}^{k}
```

Algorithm 12: METRIC_DECODE(X,ε,m,$\underline{N}$,$\mathcal{N}$): Metric based decoding algorithm for PWFA over metric spaces.

The maximum depth the algorithm is able to reach in the tree spanned by Σ is in $O(\underline{N}+\mathcal{N})$. This depth is reached when the tree traversal degenerates to a sort of depth first traversal, because there is exactly one node at depth $\underline{N}$ that has relevant children and each subsequently encountered node has exactly one relevant successor. We call this the depth first case. The minimum reached depth is in $O(\underline{N}+\log_l \mathcal{N})$ and is encountered when the tree traversal performed by the algorithm is effectively a sequence of breadth first traversals. We call this the breadth first case. The maximum number of iterations of the while loop is in $O(l\mathcal{N})$ and it occurs in the depth first case. The minimum number is in $O(\mathcal{N})$ and it arises in the breadth first case.

We assume that the metric m can be computed in $O(n)$ steps using constant memory for each pair of supplied vectors. The statement

$$R(n,l,d,\underline{N},\mathcal{N}) \in O\left(ln\mathcal{N}^2 + dn\mathcal{N} + (l^{\underline{N}} + l\mathcal{N})n^2\right) \tag{9.52}$$

holds for the total run-time $R(n,l,d,\mathcal{N})$ of the algorithm. The first term $ln\mathcal{N}^2$ describes the comparison of each vector to the set of previously seen vectors in line 10. The second term $dn\mathcal{N}$ denotes the time spent to compute instances of the product Iv in line 15. The third term $(l^{\underline{N}}+l\mathcal{N})n^2$ stems from the product $A_i v$ in line 17. In practice the run-time is often controlled by the first term, which grows quadratically in the number $\mathcal{N}$ of returned vectors.

The memory requirements of the algorithm are in $O(nl^{\underline{N}}+nl\mathcal{N})$. The first term $nl^{\underline{N}}$ arises from the number of vectors the algorithm has to enqueue to reach a depth of $\underline{N}$ in the tree spanned by Σ. The second term $nl\mathcal{N}$ describes the number of vectors it has to enqueue after having reached a depth of $\underline{N}$. It also contains the amount of space required for the storage of the set T, which is in $O(n\mathcal{N})$. In practice the memory requirements of the algorithm are mainly determined by this term $n\mathcal{N}$ and the space that is necessary for the storage of the queue can be neglected.

The algorithm is guaranteed to terminate, if $\underline{N}$ and $\mathcal{N}$ are finite. The case that $r < \underline{N}$ holds in line 12 is only possible for a finite number of iterations of the while loop. Assume that $r \geq \underline{N}$ always holds. Then the algorithm either diminishes the number of elements in the queue by one, or it adds an element to the set T in each iteration of the while loop.

If we do not limit the number of vectors produced by Algorithm 12 to some finite number (i.e. by setting $\mathcal{N} = \infty$), then it is no longer guaranteed to terminate. The automaton $X = (Q, \Sigma = \{1\}, W = (A_1), I, F)$ shown in Figure 9.12 is quite well behaved, i.e. it has the property $S(X) = T(X)$ and $S(X)$ is compact and thus bounded. The product $A_1^i F$ yields the vector $(i\ 1\ \ -i\ 1)^T$ for each $i \in \mathbb{N}$. Thus $m(A_1^i F, A_1^j F) = |A_1^i F - A_1^j F|$ is at least $|(1\ 0\ \ -1\ 0)^T| = \sqrt{2}$ for natural numbers i and j such that $i \neq j$, if we use the standard metric m. Hence the routine METRIC_DECODE never terminates, if we call it using the parameter tuple $(X, \varepsilon, m, \underline{N}, \infty)$ for $\varepsilon < \sqrt{2}$ and $\underline{N} \in \mathbb{N}$, as it successively adds the vectors $A_1^i F$ to the set T for each $i \in \mathbb{N}$. Of course all these vectors are mapped to 0, when they are multiplied by the initial matrix of the automaton.

If we substitute $m(v, v')$ by $m'(Iv, Iv')$ for some suitable metric m' in line 10 of Algorithm 12, then the modified algorithm does no longer decode all PWFA over metric spaces completely. This is shown in the following example.

Example 9.9 *Observe the automaton X shown in Figure 9.13. X computes the set* $S(X) = T(X) = \mathbb{Z}$. *After the first symbol has been consumed, the label* 2 *in X serves*

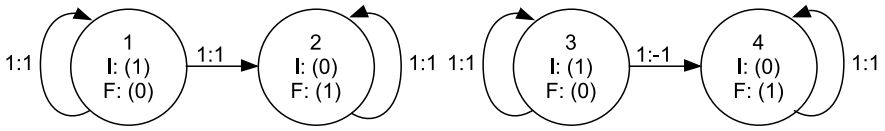


Figure 9.12: Integer PWFA X of dimension 1 over $\Sigma = \{1\}$ computing the function $f_X(w) = 0$ for each $w \in \Sigma^*$.

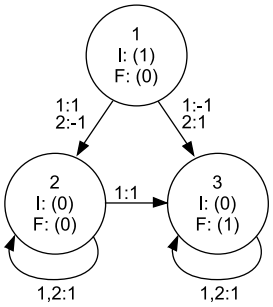


Figure 9.13: Integer PWFA X of dimension 1 over $\Sigma = \{1,2\}$ computing the set $S(X) = \mathbb{Z}$.

only the purpose of forcing the automaton to produce every vector it ever produces infinitely often (see Lemma 4.1). Let $h : \Sigma^* \mapsto \Sigma^*$ denote the homomorphism given by $h(1) = 1$ and $h(2) = \varepsilon$. Then the word function f_X of X is given by

$$f_X(w) = \begin{cases} 0 & \text{if } w = \varepsilon \\ |h(w')| - 1 & \text{if } w = 1w', \text{ for some } w' \in \Sigma^* \\ 1 - |h(w')| & \text{if } w = 2w', \text{ for some } w' \in \Sigma^* \end{cases} \tag{9.53}$$

for each $w \in \Sigma^*$. Now assume that $m(v,v')$ in line 10 of the algorithm were substituted by $m'(Iv,Iv')$ for some metric m' and we apply the modified algorithm to X for $\underline{N} = 0, \mathcal{N} = \infty$ and $\varepsilon = 0$. Algorithm 12 performs a breadth first traversal of a tree. The algorithm starts with the root of the tree, which corresponds to the empty word ε. The function $f_X(\varepsilon)$ yields 0, which the algorithm has not seen before. Thus it enqueues the nodes belonging to the words 1 and 2, which yield the values $f_X(1) = -1$ and $f_X(2) = 1$. As these also are encountered for the first time, the nodes corresponding to the words 11, 12, 21 and 22 are enqueued. The values computed for these words are $f_X(11) = 0$, $f_X(12) = -1$, $f_X(21) = 0$ and $f_X(22) = 1$. The algorithm has seen all these values before and terminates the computation, as the queue runs empty. The returned sequence is $0, -1, 1$, which is apparently incomplete, as completeness would require the algorithm to return a sequence in which every integer appears at least once.

In practice the algorithm as stated in Algorithm 12 has several problems.

One problem is that the comparison of a potentially new vector to each vector that has been inserted into the set T takes a lot of time. More precisely it causes the run-time of the algorithm to grow quadratically with the number of produced vectors. This problem can be partly remedied.

Let $X = (\{1,\ldots,n\},\Sigma,W,I,F)$ be a real PWFA of dimension d and let the metric m be defined as

$$\begin{aligned} m(v_1,v2) &= \sqrt{\tfrac{1}{n}\sum_{i=1}^{n}(v_{1,i}-v_{2,i})^2} \\ &= \sqrt{\tfrac{1}{n}}|v_1-v_2| \end{aligned} \tag{9.54}$$

for $v_1 = (v_{1,1}\ldots v_{1,n})^T, v_2 = (v_{2,1}\ldots v_{2,n})^T \in \mathbb{R}^n$, which is commonly referred to as the root mean squared error (RMSE). This is a common setting in applications. We observe that

$$\begin{aligned} & \varepsilon \leq \sqrt{\tfrac{1}{n}\sum_{i=1}^{n}(v_{1,i}-v_{2,i})^2} \\ \Leftrightarrow\ & n\varepsilon^2 \leq \sum_{i=1}^{n}(v_{1,i}-v_{2,i})^2, \end{aligned} \tag{9.55}$$

i.e. we do not have to perform the rather expensive square root operation, if we adjust the provided value ε. Further assume that the set T is not unstructured but ordered by the norm of the contained vectors, and we can find the vectors that have a norm contained in a certain interval quickly. This can in practice be implemented using for instance an ordered and balanced binary tree (e.g. a red black tree, cf. [15]) to represent T, where the keys are the vector norms. The implementation has to allow that the same key occurs multiple times in the data structure, because various vectors having the same norm may be included in T at the same time. A concrete example of such a data structure is the multimap template class found in the C++ standard library. Inserting a key and searching for a key in such an ordered and balanced binary tree both take $O(\log_2|T|)$ time, i.e. the growth is logarithmic in the number of keys already present. Now assume that we are considering a potentially new vector v with norm $|v|$. Let $\delta \geq 1$ denote a real number. The minimum RMSE between v and a vector v' such that $|v'| = \delta|v|$ is $\sqrt{\frac{1}{n}}(\delta-1)|v|$. This implies

$$\delta \geq \frac{\varepsilon\sqrt{n}}{|v|} + 1 \tag{9.56}$$

for $v \neq 0$. If v equals 0, then we have to check all vectors v' that have a norm of at most $\sqrt{n}\varepsilon$. The case for $\delta < 1$ is obtained analogously. We can thus compute an interval of relevant norms and disregard all vectors stored in T that possess norms lying outside this interval. In practice this often reduces the number of vectors that have to be checked to a small fraction of all vectors that are contained in T. In the worst case we still have to compare each current vector to each previously seen vector.

A second way to reduce the number of vectors, a potentially new vector has to be compared to, is to divide the set T into a finite set of sets T_i, where $1 \leq i \leq q$ for some $q \in \mathbb{N}$. Some hash function (cf. [15]) is used to assign new vectors to these sets. Let $r \in \mathbb{R}^n$ such that $|r| = 1$. A hash function $H(v) : \mathbb{R}^n \mapsto \{1,\ldots,q\}$ that works well in practice is the following:

1. Compute the cosine of the angle φ between the current vector v and r using the formula
$$\cos(\varphi) = \frac{vr}{|v|} \ . \tag{9.57}$$
2. Compute the hash value $H(v) = \left\lfloor \left(\frac{(\cos(\varphi)+1)(q-1)}{2} + 1 \right) + \frac{1}{2} \right\rfloor$.

This approach however changes the output of the algorithm. The set $\mathcal{F}$ defined in Algorithm 12 no longer contains only vectors that have disjoint ε neighborhoods, as vectors with overlapping neighborhoods may be assigned to different sets T_i and T_j for $i \neq j$ by the hash function. The algorithm still completely decodes each real PWFA though.

A second problem of the algorithm consists in its high memory requirements. These can possibly be reduced by means of standard data compression methods, but we have not conducted experiments in this direction. We conjecture that storing the set T in compressed form would severely degrade the performance of the algorithm.

Another problem is a good choice of the metric m. The RMSE metric is often used in practice. Each vector component (and thus each state) is given the same significance by this metric. In practice we often encounter cases, where different states of a PWFA do not have the same significance, as the following example shows.

Example 9.10 *Consider a real PWFA X of dimension 3 representing a bi-level movie of a rotating object, where $S(X) = T(X)$ holds. Assume that the first two components x and y of each vector computed by f_X describe a spatial position and take values in the interval $[1,600]$ (i.e. each quantized movie frame is interpreted as a 600×600 pixel image) and that the third component corresponds to a temporal position in $\mathbb{R}$, which corresponds to the rotation angle (i.e. the object is rotated by an angle of 1 at the time index 1). Further assume without loss of generality that each dimension of X works on a separate set of states. One complete rotation of the object is contained in a subset of $R = [1,600] \times [1,600] \times [0,2\pi]$. Assume that X is in initial normal form and each non-zero element of the initial matrix is 1. Then the range R is also reflected in the set of vectors $\{\mu_X(w)F_X | w \in \Sigma_X^*\}$. If we use the metric based decoding algorithm to decode X, where m is chosen as the RMSE metric, then clearly the spatial components are overemphasized in comparison to the temporal component. This leads to a bad decoding performance. We do however not need to modify the metric in this case. It suffices to multiply the final weights of the states computing the temporal component by some number $s > 1$ and the initial weights corresponding to the component by $1/s$. This does not change the word function computed by X, but it puts more emphasis on the time component.*

9.2.6 Comparison of decoding algorithms

We compare the decoding algorithms presented above. Some, specifically the stack, queue and chaos game based algorithms, turn out be useless in applications, as they perform worse than other algorithms in every practical setting.

In the case of the stack and queue based algorithms, the reason is that they show bad run-time performance when compared to the vector caching approach.

The chaos game algorithm is unacceptable in many applications, because it is probabilistic. Consider for instance the situation that we want to render the curve represented by the automaton shown Figure 4.2 and subsequently fill it using a flood fill algorithm. Then there is always some non-zero probability that the curve is incomplete and the filling algorithm also fills areas outside the curve. The situation that an area delimited by spline curves is subsequently filled is very common in character rendering.

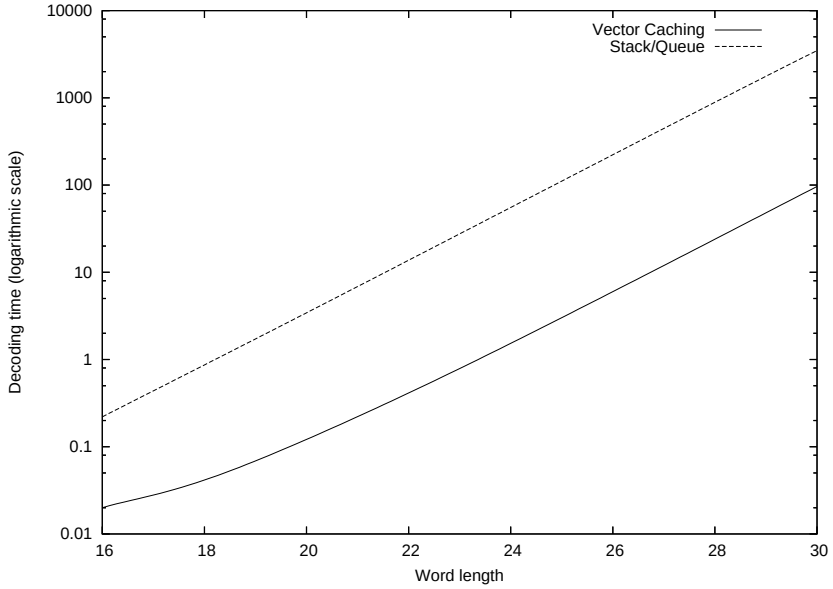


Figure 9.14: The graph shows the decoding time used for the curve automaton shown in Figure 4.2 plotted against the word length for implementations of the stack based, queue based and vector caching based algorithms. As the stack based and queue based algorithms have basically the same run-time, we have merged their curves to a single curve.

This leaves the vector caching and metric based algorithms. Comparing these, there is no single best algorithm, i.e. there are automata which are very efficiently decoded by the caching based algorithm, but provoke a very poor behavior of the metric based algorithm and vice versa.

The run-times provided in this subsection were all measured on an Intel Core 2 CPU running at 2.4 GHz and having 4GB of random access memory. The hardware clock used to measure the run-times had a resolution of one hundredth of a second.

Comparison of the word length based decoding algorithms

The stack, queue and vector caching based decoding algorithms are similar in the sense that they all decode a given PWFA X by evaluating the $f_X(w)$ for all words up to a certain given word length $\overline{N}$. Figure 9.14 shows the decoding times obtained by applying implementations of these algorithms to the curve automaton shown in Figure 4.2. The automaton was decoded for the word lengths 16 to 30. The graph was obtained by interpolating the obtained values using Bézier curves. As the run-time of the vector caching based algorithm for word lengths smaller than 16 was too small to obtain any meaningful measurements, we have not plotted the graph for such word lengths. As expected, the decoding time using the queue and stack based algorithms is approximately the same. Obviously, the vector caching based algorithm outperforms the stack and queue based algorithms by far. The distance becomes even larger, if we

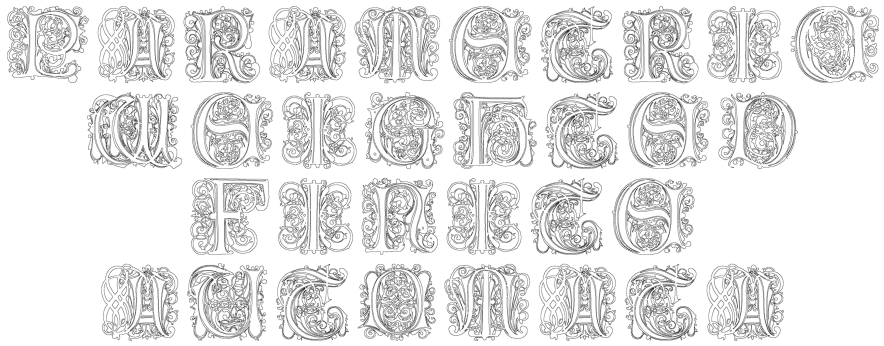


Figure 9.15: Decoded image of a PWFA computing the outlines of the text *Parametric Weighted Finite Automata* set in the *Paulus Franck Initialen* font.

consider larger automata, especially when they are unbalanced (see subsection 9.2.3). The curve automaton in Figure 4.2 has only 4 states and we have successfully used the vector caching based algorithm to decode automata having in the order of 100000 states. Meaningful automata of this size can easily be obtained by displaying texts using sophisticated letters. Figure 9.15 shows an image generated by a 81394 state PWFA representing 40687 cubic Bézier curves. The automaton was decoded in 80 minutes using a word length of 24. The estimated decoding time using the stack based algorithm is roughly 16 days. But the decoding time of the caching based algorithm is also very high. If we decode each single letter for itself (where each occurrence is counted, i.e. the letter a is decoded 5 times) and sum up the times, then the accumulated time is only 4.63 seconds. This shows that there is still a large potential for optimization in the algorithm. In the special case of this automaton, a large speed up could be gained by partitioning the computation using the method given in Theorem 5.2.

In summary this means that the stack and queue based algorithms are not used in practice. The higher memory consumption of the caching based algorithm can usually be neglected. Assume that we would want to decode a certain automaton for a certain word length and the amount of memory required for the caches could barely be satisfied by the memory found in a contemporary computer. Then this would most probably imply that the run-time of the algorithm would be intolerably high.

Comparison of the metric and vector caching based decoding algorithms

The caching based algorithm is a good choice, if we want to decode a PWFA, in which each label is equally important. This holds e.g. in the case of PWFA computing real polynomials and automata obtained by operations on polynomials, e.g. splines and spline patches. An image rendered by an instance of such an automaton is shown in Figure 9.15. The metric based algorithm performs at least as poor as the stack and queue based algorithms on this automaton.

One example, where the labels are not equally important, is the automaton depicted in Figure 6.1, which computes the fractal fern. Observing Figure 6.2, we see that the area filled by the subset $w_2\mathcal{A}$ is much bigger than that filled


Figure 9.16: The two images were obtained by decoding the automaton shown in Figure 6.1 as a bi-level image. The left image was produced using the vector caching based algorithm using prefix and suffix caching, where the automaton was decoded for all words in $\{1,2,3,4\}^{18}$. The image on the right was produced by the metric based algorithm for $\mathcal{N} = \infty$, $\underline{N} = 0$ and $\varepsilon = 10^{-3}$, where the maximal encountered word length was 129.

by all the other transformed subsets of the attractor. Figure 9.16 compares the performance of the metric and vector caching based algorithms on the fern. The measured run-time of the caching based algorithm for all words of length 18 was roughly 100 minutes. If we call the metric based algorithm with the parameters $\varepsilon = 10^{-3}$ and $\mathcal{N} = \infty$, then the algorithm terminates after 7 minutes and 21 seconds. The maximal reached depth in the tree spanned by the alphabet $\{1,2,3,4\}$ is 129, which clearly is far beyond the word length 18 and any other word length that can effectively be handled by the vector caching based algorithm.

The vector caching based decoding algorithm almost always performs very badly on any PWFA computing an iterated image of a non-trivial PWFA computable set. Consider the automaton X' given in Figure 9.3. If we interpret X' as an RGB image, we obtain the image shown in Plate II in the appendix. X' itself is decoded perfectly well by the vector caching based algorithm, as both labels are equally important. Let X be the automaton described in Example 9.4, which computes an iterated image of the set $S(X')$. The interpretation of $S(X)$ as an RGB image is shown in Plate IV. X is decoded very poorly by the caching based algorithm. The image was decoded using the metric based algorithm, where the maximal encountered word length was 72. The problems of the caching based algorithm are caused by various reasons. One major reason is that the splitting label 3 introduced by the iterated image construction introduces a massive amount of redundancy in the computation, i.e. if α and γ are words in $\{1,2\}^*$, then $f_X(\alpha\beta\gamma)$ yields the same vector for each $\beta \in 33^*$. Another reason is that the construction often produces automata Y such that important portions of $T(Y)$ are only observed for word lengths that are too high to be handled by implementations of the caching base algorithm. Consider Example 9.7, where a PWFA X representing an RGB movie of a rotating fractal is presented. If we want to decode X for the purpose of rendering a movie consisting of k frames, then the required word length is at least k. This implies that the

decoding of a two second movie at 25 frames per second cannot be handled by the vector caching algorithm, as the run-time would be at least in the order of weeks.

Chapter 10

Conclusion

In this thesis we have studied the theory and applications of parametric weighted finite automata. We have shown that PWFA are very powerful devices concerning their theoretical as well as practical aspects. Concerning the theoretical aspects of PWFA this goes so far that some important problems, including the membership, emptiness and equivalence problem, are recursively undecidable. These results also imply the non-existence of an effective state minimization algorithm for PWFA. We can however apply some forms of syntactical simplifications to PWFA. One such simplification is that each PWFA computable set can be computed by an automaton that has an alphabet cardinality of at most two. Another simplification is that for each PWFA there are equivalent PWFA that either have only a single initial state per dimension or a single final state. Concerning the word function computed by a PWFA, we can always construct an WFA over a different algebraic structure that computes the same function. Thus the real difference between the discussion of plain WFA and PWFA is that for a PWFA X we consider the set $S(X)$ instead of only the function f_X. If two subsets of a set are PWFA computable, then this also holds for their union. The set of practically relevant PWFA is closed under affine transformation. If a PWFA X computing a set $S(X)$ satisfies the condition that $S(X) = T(X)$, then each iterated image of $S(X)$ is PWFA computable. We have shown that some properties of transitions matrices can be checked algorithmically. One such important property is the contractivity of a component rational complex matrix. We have discussed the relation of PWFA to IFS, RIFS and MRFS, where we have provided methods for transforming IFS, RIFS and MRFS into equivalent PWFA. We have shown that PWFA can be used to represent a large variety of functions. Some real functions are computable by PWFA on finite intervals. This includes polynomials, scaling functions an wavelets. PWFA can also be used to compute interval evaluations of real polynomials on the real unit interval. Some real and complex functions, including some real and complex rational functions, real and complex polynomials and the complex exponential function, can be computed by PWFA on their complete domain.

Based on the theoretical results above, we have also presented numerous PWFA applications. PWFA can be used to represent attractors of IFS, RIFS and MRFS. The automata constructed in WFA based image compression are also valid PWFA, thus we can represent pixel images as PWFA. Bézier curves, Catmull-Rom splines and B-splines as well as the resulting spline patches can be

represented by PWFA. Bézier curves are used in many applications, including font design and CAD. Bézier patches can be used for 3d modeling of real objects. Catmull-Rom splines are very popular in image scaling applications. B-splines allow the construction of interpolating curves for every degree of curve continuity.

We have presented ways to interpret PWFA computed sets as images, 3d scenes, polygonal surfaces and movies. Finally, we have shown that these interpretations can be efficiently computed by providing several PWFA decoding algorithms.

A collection of some interesting open problems of PWFA theory is given below.

Problem 10.1 *Is the set $\mathcal{D}(\mathbb{R})$ closed under the set intersection operation?*

Problem 10.2 *If the answer to Problem 10.1 is positive, is the closure effective, i.e. given two real PWFA X and Y, is there an effective algorithm constructing a PWFA Z such that $S(Z) = S(X) \cap S(Y)$?*

Problem 10.3 *Is there a PWFA that computes the exponential, sine, cosine or any analytic real function f, such that each derivative of f is not the constant zero function, on a compact subset of $\mathbb{R}$ containing an infinite number of elements?*

If the answer to Problem 10.1 is positive, then the answer to Problem 10.3 is also positive, as we could e.g. compute the real exponential function on the real unit interval (see subsection 7.2.2).

Problem 10.4 *Do ST-consistent automata X exist, such that $S(X)$ is not computable by a PWFA Y such that $S(Y) = T(Y)$?*

Problem 10.5 *Are there PWFA computable sets $S \neq \emptyset$ such that there is no ST-consistent PWFA X computing $S(X) = S$?*

Problem 10.6 *Is the membership problem for automata in the class cPWFA decidable?*

Problem 10.7 *Is every real WFA computable function also PWFA computable?*

We have demonstrated in Remark 7.1 that it is only trivial to construct a PWFA X computing a real function defined by a WFA Y, if f_Y is total.

Problem 10.8 *Is the set of sets computable by unary alphabet PWFA X over $\mathbb{R}$, for which $S(X) = T(X)$ holds, effectively closed under iterated images, i.e. given an arbitrary unary alphabet PWFA of dimension $d \in \mathbb{N}^+$ over $\mathbb{R}$ such that $S(X) = T(X)$ and an arbitrary finite sequence $g_1, \ldots, g_k$ of affine transformations on $\mathbb{R}^d$, does there always exist a unary alphabet PWFA Y of dimension d over $\mathbb{R}$ computing the iterated image of $S(X)$ under $g_1, \ldots, g_k$?*

A very interesting open problem in PWFA application consists of the construction of a good algorithm for PWFA inference, i.e. an algorithm that given a positive real number ε and a compact subset S of a metric space $(\mathcal{S}^d, m)$, where $d \in \mathbb{N}^+$ and $\mathcal{S}$ is a semiring, constructs a small PWFA X of dimension d over $\mathcal{S}$ such that $h(m)(S(X), S) < \varepsilon$. This problem however is likely at least as difficult to solve as the construction of a good algorithm solving the well known fractal inverse problem (cf. [49]).

Appendix A - Color Plates

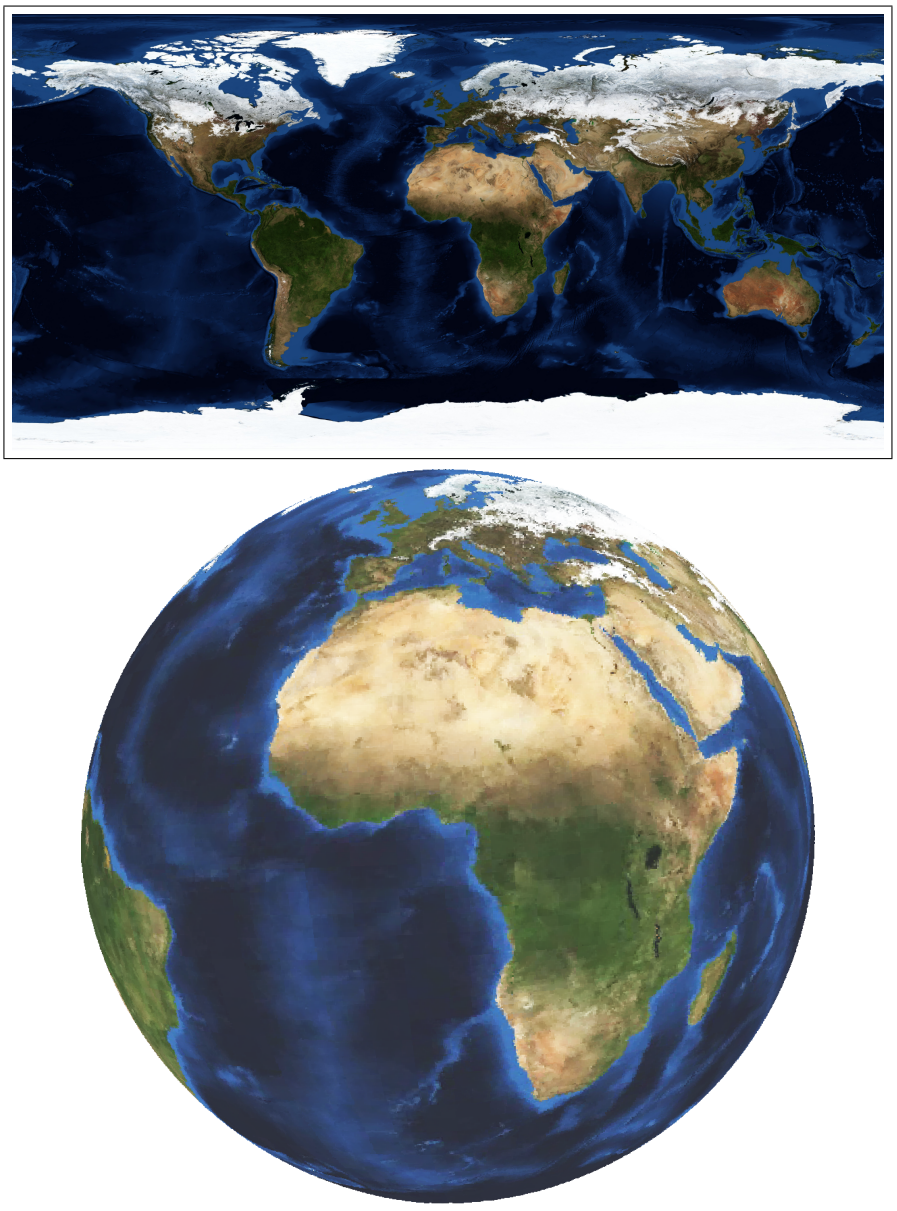


Plate I: The image of the earth on top (taken from the NASA - Visible Earth website [77]) was WFA compressed and used to texture an approximated sphere built from 8 bicubic Bézier patches, which is shown at the bottom.

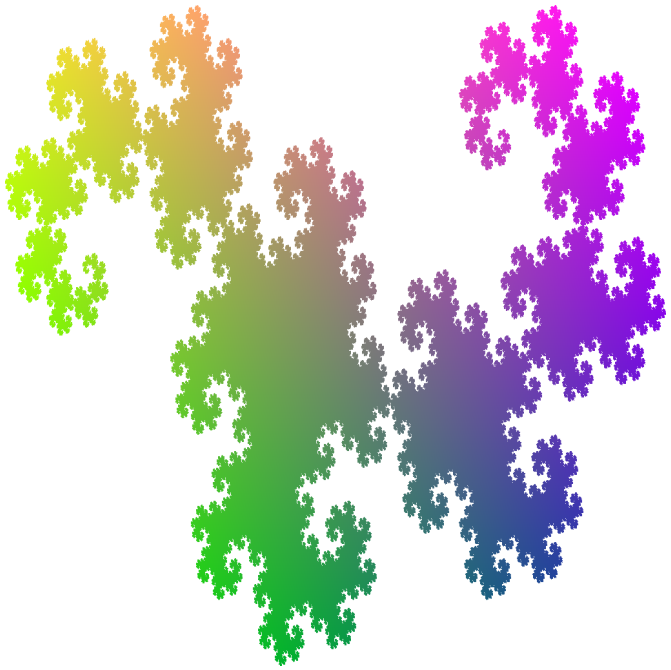


Plate II: Colored fractal dragon obtained by interpreting the automaton shown in Figure 9.3 as an RGB color image.

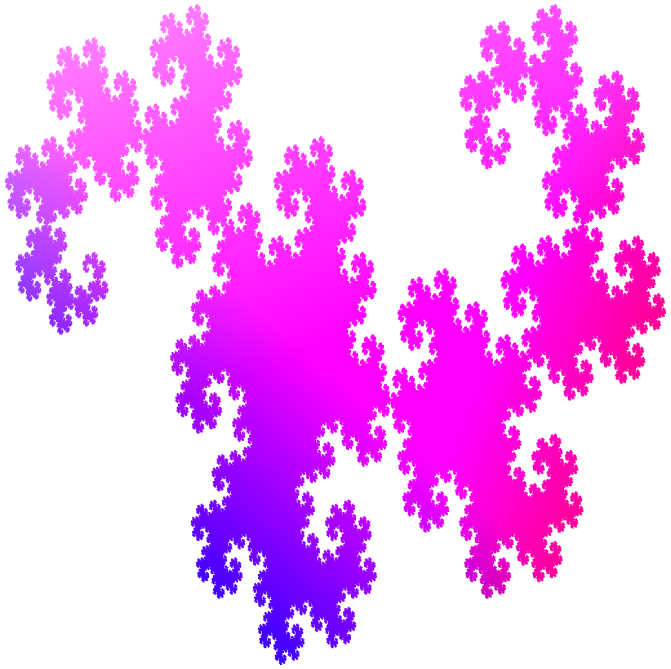


Plate III: Colored fractal dragon obtained by interpreting the automaton shown in Figure 9.3 as a YCbCr color image.

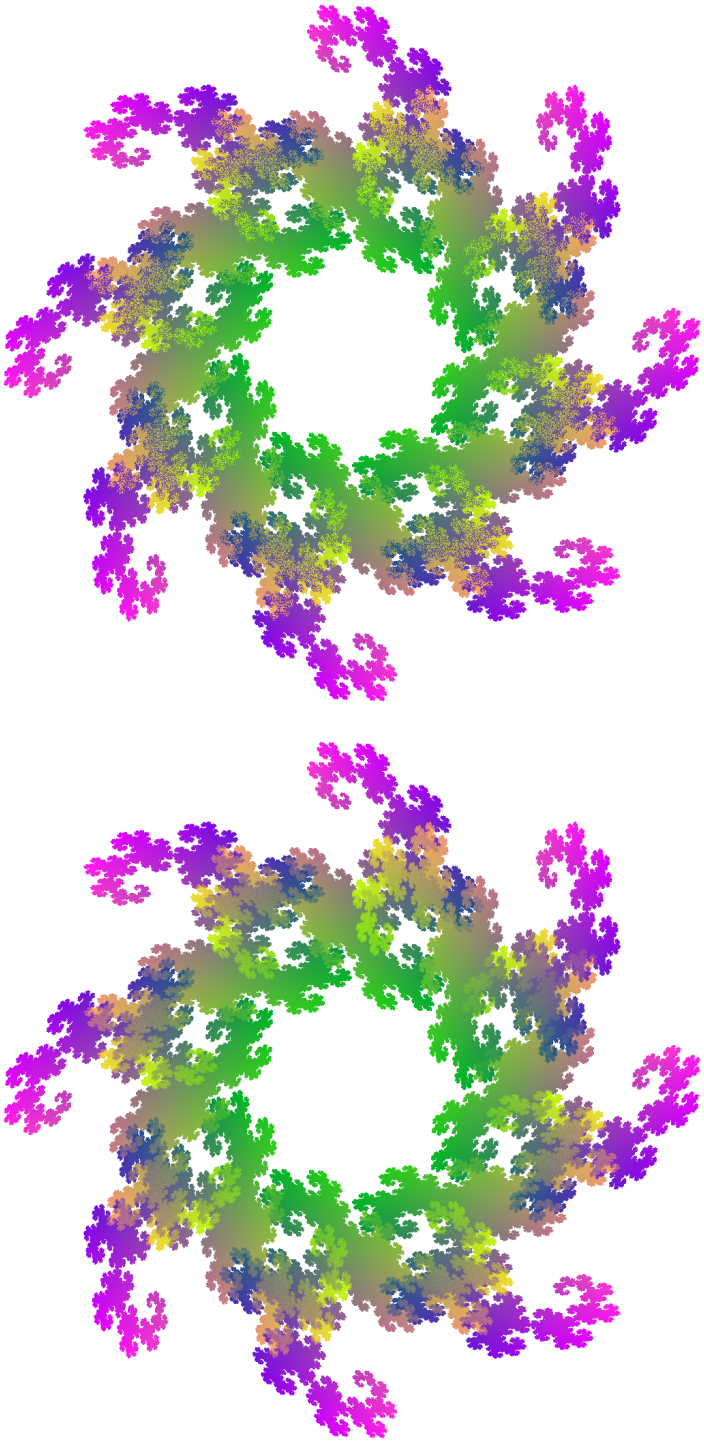


Plate IV: Let X' denote the automaton shown in Figure 9.3 and let X the PWFA obtained by applying an iterated affine transformation representing a rotation by $\pi/4$ to X'. The image on top and bottom show the interpretation of X as an RGB image without and with the usage of an averaging buffer respectively.

Plate V: The set computed by the automaton shown in Figure 9.8 is augmented by adding three components R, G and B to each vector $(x\ y\ z)^T$ that are assigned the values $\frac{x+y+2}{4}$, $\frac{x+z+2}{4}$ and $\frac{y+z+2}{4}$ respectively. The image shows the interpretation of the augmented set as an RGB 3d scene rendered from a point on the positive z-axis for $z > 1$ looking at the origin of $\mathbb{R}^3$.

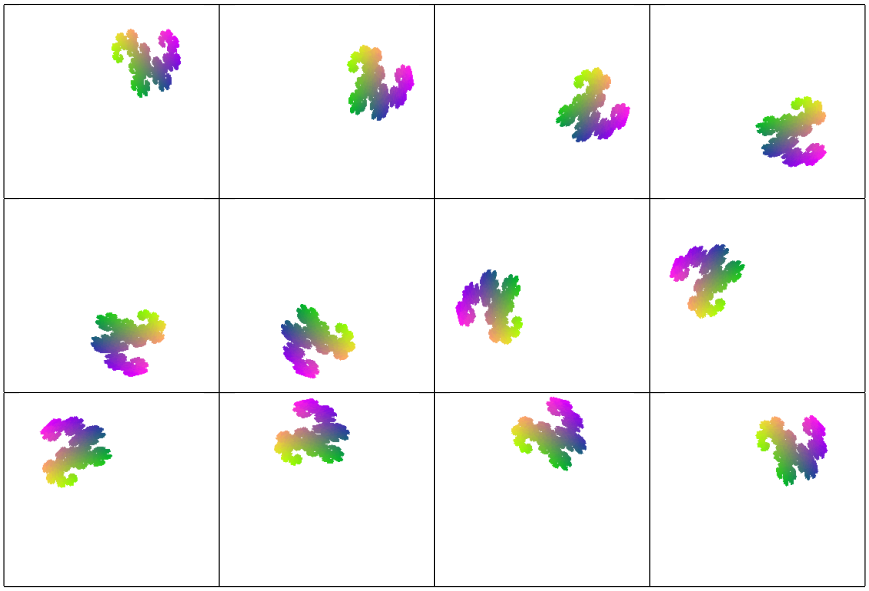


Plate VI: A PWFA representing a rotating colored fractal dragon is interpreted as an RGB movie. The automaton computes a single dragon for each time index and each time index corresponds to a certain rotation angle. The single images show frames of this movie.

Bibliography

[1] Adobe Systems Inc. *Adobe Type 1 Font Format*. Addison Wesley, 2nd edition, 1990.

[2] Adobe Systems Inc. *PostScript Language Reference*. Addison-Wesley, 3rd edition, 1999.

[3] Adobe Systems Inc. *PDF Reference Version 1.6*. Adobe Press, 5th edition, 2004.

[4] J. Albert and J. Kari. Parametric weighted finite automata and iterated function systems. In *Proceedings of the Conference Fractals in Engineering*, pages 248–255, Delft, 1999.

[5] G. Alefeld and J. Herzberger. *Introduction to Interval Computations*. Academic Press, 1983.

[6] Anonymous. English Wikipedia article "Typeface". `http://en.wikipedia.org/wiki/Typeface`, 2007. Visited on August 22th 2007.

[7] M. J. Aramini. Efficient image magnification by bicubic spline interpolation. `http://mysite.verizon.net/~vze2vrva/design.html`. Visited on April 18th 2007.

[8] M. F. Barnsley. *Fractals everywhere*. Morgan Kaufmann, 2nd edition, 2000.

[9] M. F. Barnsley, J.H. Elton, and D. P. Hardin. Recurrent iterated function systems. *Constructive Approximation*, 5:3–31, 1989.

[10] J. Berstel and C. Reutenauer. *Rational Series and Their Languages*. Springer-Verlag, 1988.

[11] W. Böhm. Generating the Bézier points of B-spline curves and surfaces. *Computer Aided Design*, 13(6):365–366, 1981.

[12] S. Bozapalidis and A. Grammatikopoulou. Recognizable picture series. *Journal of Automata, Languages and Combinatorics*, 10(2/3):159–183, 2005.

[13] E. Catmull and R. Rom. A class of local interpolating splines. In R. Barnhill and R. Riesenfeld, editors, *Computer Aided Geometric Design*, pages 317–326. Academic Press, 1974.

[14] A. Cohen, I. Daubechies, and J. C. Feauveau. Biorthogonal bases of compactly supported wavelets. *Communications on Pure and Applied Mathematics*, 45(5):485–560, 1992.

[15] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms*. MIT Press, 1990.

[16] K. Culik II and S. Dube. Affine automata: A technique to generate complex images. In *Mathematical Foundations of Computer Science 1990*, volume 452 of *Lecture Notes Computer Science*, pages 224–231. Springer-Verlag, 1990.

[17] K. Culik II and S. Dube. Affine automata and related techniques for generation of complex images. *Theoretical Computer Science*, 116(2):373–398, 1993.

[18] K. Culik, II and S. Dube. L-systems and mutually recursive function systems. *Acta Informatica*, 30:279–302, 1993.

[19] K. Culik II and S. Dube. Implementing Daubechies wavelet transform with weighted finite automata. *Acta Informatica*, 34(5):347–366, 1997.

[20] K. Culik II and J. Karhumäki. Finite automata computing real functions. *SIAM Journal on Computing*, 23(4):789–814, 1994.

[21] K. Culik II and J. Kari. Image compression using weighted finite automata. *Computers & Graphics*, 17(3):305–314, 1993.

[22] K. Culik II and J. Kari. Mechanisms for pattern generation. *Complex Systems*, 7:347–365, 1993.

[23] K. Culik II and J. Kari. Parametrized recurrent systems for image generation. *Information Processing Letters*, 48(6):267–274, 1993.

[24] K. Culik II and J. Kari. Image-data compression using edge-optimizing algorithm for WFA inference. *Information Processing and Management*, 30(6):829–838, 1994.

[25] K. Culik II and J. Kari. On the power of L-systems in image generation. *Acta Informatica*, 31:761–773, 1994.

[26] K. Culik II and J. Kari. Inference algorithms for WFA and image compression. In Y. Fisher, editor, *Fractal Image Compression: Theory and Application*. Springer-Verlag, 1995.

[27] K. Culik II and J. Kari. Finite state transformation of images. *Computers & Graphics*, 20(1):125–135, 1996.

[28] K. Culik II and J. Kari. Computational fractal geometry with WFA. *Acta Informatica*, 34(2):151–166, 1997.

[29] K. Culik II, V. Valenta, and J. Kari. Compression of silhouette-like images based on WFA. *Journal of Universal Computer Science*, 3(10):1100–1113, 1997.

[30] K. Culik II and P. C. von Rosenberg. Generalized weighted finite automata based image compression. *Journal of Universal Computer Science*, 5(4):227–242, 1999.

[31] C. W. Curtis. *Linear Algebra: An Introductory Approach*. Springer-Verlag, corrected fourth printing edition, 1993.

[32] I. Daubechies. Orthonormal bases of compactly supported wavelets. *Communications on Pure and Applied Mathematics*, 41:909–996, 1988.

[33] C. de Boor. *A practical guide to splines*. Springer-Verlag, 1978.

[34] D. Derencourt, J. Karhumäki, M. Latteux, and A. Terlutte. On computational power of weighted finite automata. In Ivan M. Havel and Vaclav Koubek, editors, *Proceedings of Mathematical Foundations of Computer Science '92*, volume 629 of *LNCS*, pages 236–245. Springer-Verlag, 1992.

[35] D. Derencourt, J. Karhumäki, M. Latteux, and A. Terlutte. On continuous functions computed by finite automata. *RAIRO - Informatique théoretique et Applications*, 28(3-4):387–403, 1994.

[36] M. Droste, J. Kari, and P. Steinby. Observations on the smoothness properties of real functions computed by weighted finite automata. *Fundamenta Informaticae*, 73(1,2):99–106, 2006.

[37] R. Dunlop. Introduction to Catmull-Rom Splines. `http://www.mvps.org/directx/articles/catmull/`. Visited on April 18th 2007.

[38] S. Eilenberg. *Automata, Languages and Machines*, volume A. Academic Press, New York, 1974.

[39] J. Eisner. Simpler and more general minimization for weighted finite-state automata. In M. Hearst and M. Ostendorf, editors, *Human Language Technology conference - North American chapter of the Association for Computational Linguistics 2003 (HLT-NAACL 2003): Main Proceedings*, pages 64–71. Association for Computational Linguistics, 2003.

[40] M. Eramian. *Image Texture Analysis using Weighted Finite Automata*. Doctoral dissertation, University of Western Ontario, London, Ontario, 2002.

[41] G. Farin. *Curves and Surfaces for Computer Aided Geometric Design*. Academic Press, 1990.

[42] J. Ferraiolo, J. Fujisawa, and D. Jackson. Scalable vector graphics (SVG) 1.1 specification. World Wide Web Consortium, Recommendation REC-SVG11-20030114, 2003.

[43] L. Fries. *Histori, Namen, Geschlecht, Wesen, Thaten, gantz leben und sterben der gewesnen Bischoffen zu Wirtzburg und Hertzogen in Franken, Auch was bey einem Jeden in Zeit seiner Regirung, sunderlich gehandelt worden, ergangen vnd beschehen ist*. Handschrift Universitätsbibliothek Würzburg, M.ch.f.760, 1582.

[44] L. Fries. Chronik der Bischöfe von Würzburg - Die Prachthandschrift des Fürstbischofs Julius Echter als Multimedia-DVD. Universitätsbibliothek Würzburg, 2004.

[45] M. Giraud and D. Lavenier. Linear encoding scheme for weighted finite automata. In M. Domaratzki, A. Okhotin, K. Salomaa, and S. Yu, editors, *CIAA 2004*, volume 3317 of *Lecture Notes in Computer Science*, pages 146–155. Springer-Verlag, 2004.

[46] M. Giraud and D. Lavenier. Dealing with hardware space limits when removing epsilon-transitions in a genomic weighted finite automaton. *Journal of Automata, Languages and Combinatorics*, 10(2/3):265–285, 2005.

[47] A. Grammatikopoulou. Prefix picture sets and picture codes. In S. Bozapalidis, A. Kalampakas, and G. Rahonis, editors, *Proceedings of the 1st International Conference on Algebraic Informatics (CAI 2005)*, Aristotle University of Thessaloniki, pages 255–268, 2005.

[48] A. Graps. An introduction to wavelets. *IEEE Computational Science and Engineering*, 2(2):50–61, 1995.

[49] É. Guerin and É. Tosan. Fractal inverse problem: Approximation formulation and differential methods. In J. Lévy-Véhel and E. Lutton, editors, *Fractals in Engineering: New Trends in Theory and Applications*, pages 271–285. Springer-Verlag, 2005.

[50] U. Hafner. *Low Bit-Rate Image and Video Coding with Weighted Finite Automata*. Doctoral dissertation, University of Würzburg, Germany, 1999.

[51] V. Halava. Decidable and undecidable problems in matrix theory. Technical Report 127, Turku Centre for Computer Science, 1997.

[52] V. Halava and T. Harju. Languages accepted by integer weighted finite automata. In J. Karhumäki, H. A. Maurer, G. Paun, and G. Rozenberg, editors, *Jewels are Forever*, pages 123–134. Springer, 1999.

[53] V. Halava and T. Harju. Undecidability in integer weighted finite automata. *Fundamenta Informaticae*, 38(1-2):189–200, 1999.

[54] V. Halava, T. Harju, M. Hirvensalo, and J. Karhumäki. Skolem's problem - on the border between decidability and undecidability. Technical Report 683, Turku Centre for Computer Science, 2005.

[55] J. C. Hart. *Computer Display of Linear Fractal Surfaces*. Doctoral dissertation, University of Illinois, Chicago, 1991.

[56] J. C. Hart. Iterated Function Systems and Recurrent Iterated Function Systems. `http://graphics.cs.uiuc.edu/~jch/procgraph/phd-notes.pdf`, 1996. Visited on January 24th 2008.

[57] M. D. Hash. A complete algorithm for real-time spline-based character animation. `http://www.ibiblio.org/e-notes/Hash/Hash.htm`. Visited on August 25 2007.

[58] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Reading, Massachusetts, 1979.

[59] R. A. Horn and C. R. Johnson. *Matrix analysis*. Cambridge University Press, 1990.

[60] J. E. Hutchinson. Fractals and self-similarity. *Indiana University Mathematics Journal*, 30(5):713–747, 1981.

[61] K. Joy. Catmull-Rom Splines. `http://graphics.cs.ucdavis.edu/education/CAGDNotes/Catmull-Rom-Spline/Catmull-Rom-Spline.html`. Visited on April 18th 2007.

[62] J. Karhumäki, W. Plandowski, and W. Rytter. Pattern-matching problems for 2-dimensional images described by finite automata. Technical Report 58, Turku Centre for Computer Science, 1996.

[63] J. Kari and P. Fränti. Arithmetic coding of weighted finite automata. *RAIRO - Informatique théoretique et Applications*, 28(3-4):343–360, 1994.

[64] F. Katritzke. *Refinements of Data Compression Using Weighted Finite Automata*. Doctoral dissertation, University of Siegen, Germany, 2001.

[65] F. Katritzke, W. Merzenich, and M. Thomas. Enhancements of partitioning techniques for image compression using weighted finite automata. *Theoretical Computer Science*, 313(1):133–144, 2004.

[66] K. Kuratowski. *Topology*, volume I. Academic Press, 1966.

[67] O. S. Lawlor and J. C. Hart. Bounding recursive procedural models using convex optimization. In *Proceedings of the 11th Pacific Conference on Computer Graphics and Applications (PG'03)*, pages 283–292. IEEE Computer Society, 2003.

[68] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. *ACM Computer Graphics*, 21(4):163–169, 1987.

[69] L. Maisonobe. Drawing an elliptical arc using polylines, quadratic or cubic Bézier curves. `http://www.spaceroots.org/documents/ellipse/index.html`, 2007. Visited on March 14th 2007.

[70] S. Mallat. Multiresolution approximation and wavelets. *Transactions of American Mathematical Society*, 315:69–88, 1989.

[71] S. Mallat. *A Wavelet Tour of Signal Processing*. Academic Press, 1998.

[72] Z. Manna. *Mathematical Theory of Computation*. McGraw-Hill, New York, 1974.

[73] I. Mäurer. Recognizable and rational picture series. In S. Bozapalidis, A. Kalampakas, and G. Rahonis, editors, *Proceedings of the 1st International Conference on Algebraic Informatics (CAI 2005)*, Aristotle University of Thessaloniki, pages 141–155, 2005.

[74] Microsoft Corporation. Truetype specifications. http://www.microsoft.com/typography/specs/default.htm, 2003. Visited on August 22nd 2007.

[75] M. Mohri. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(4), 1997.

[76] J. R. Munkres. *Topology*. Prentice-Hall, 2000.

[77] NASA. Blue Marble Next Generation. http://visibleearth.nasa.gov/view_rec.php?id=7100, 2005. Visited on September 4th 2007.

[78] M. S. Paterson. Unsolvability in 3 x 3 matrices. *Studies in Applied Mathematics*, 49:105–107, 1970.

[79] T. Pavlidis. *Algorithms for Graphics and Image Processing*. Computer Science Press, 1982.

[80] E. L. Post. A variant of a recursively unsolvable problem. *Bulletin of the American Mathematical Society*, 52:264–268, 1946.

[81] M. P. Schützenberger. On the definition of a family of automata. *Information and Control*, 4(2–3):245–270, 1961.

[82] P. Selinger. Potrace: a polygon-based tracing algorithm. http://potrace.sourceforge.net/potrace.pdf, 2003. Visited on August 3rd 2007.

[83] J. Shallit and J. Stolfi. Two methods for generating fractals. *Computers and Graphics*, 13(2):185–191, 1989.

[84] C. E. Shannon and J. McCarthy. *Automata Studies. (AM-34) (Annals of Mathematics Studies)*. Princeton University Press, 1956.

[85] L. Staiger. ω-Languages. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume III, pages 339–387. Springer-Verlag, 1997.

[86] G. A. Stanislav. Drawing a circle with Bézier curves. http://www.whizkidtech.redprince.net/bezier/circle/, 2007. Visited on March 12th 2007.

[87] D. S. Taubman and M. W. Marcellin. *JPEG2000 Image Compression Fundamentals, Standards and Practice*. Kluwer Academic Publishers, 2002.

[88] G. Tischler. Parametric weighted finite automata for figure drawing. In M. Domaratzki, A. Okhotin, K. Salomaa, and S. Yu, editors, *CIAA 2004*, volume 3317 of *Lecture Notes in Computer Science*, pages 259–268. Springer-Verlag, 2004.

[89] G. Tischler. Properties and applications of parametric weighted finite automata. *Journal of Automata, Languages and Combinatorics*, 10(2/3):347–365, 2005.

[90] G. Tischler. Real functions computed by parametric weighted finite automata. In S. Bozapalidis, A. Kalampakas, and G. Rahonis, editors, *Proceedings of the 1st International Conference on Algebraic Informatics (CAI 2005)*, Aristotle University of Thessaloniki, pages 129–138, 2005.

[91] G. Tischler. Refinement of near random access video coding with weighted finite automata. In O. H. Ibarra and H.-C. Yen, editors, *CIAA 2006*, volume 4094 of *Lecture Notes in Computer Science*, pages 46–57. Springer-Verlag, 2006.

[92] G. Tischler. On Computability and some Decision Problems of Parametric Weighted Finite Automata. *Journal of Automata, Languages and Combinatorics*, 12(4):525–544, 2007.

[93] G. Tischler, J. Albert, and J. Kari. Parametric weighted finite automata and multidimensional dyadic wavelets. In *Proceedings of the Conference Fractals in Engineering 2005*, Tours, 2005. Published on CD.

[94] G. Tischler and J. Wolff von Gudenberg. Solving decidability problems with interval arithmetic, 2008. To be submitted.

[95] F. von Haeseler, H.-O. Peitgen, and G. Skordev. Cellular automata, matrix substitutions and fractals. *Annals of Mathematics and Artificial Intelligence*, 8(3-4):345–362, 1993.

[96] A. Watt. *3D Computer Graphics*. Addison-Wesley, 3rd edition, 2000.

[97] G. Williams. FontForge. `http://fontforge.sourceforge.net/`, 2007. Visited on August 22 2007.

Picture credits

I am grateful to the following persons and/or institutions for allowing me to reproduce the following images:

- **Martin von Wagner Museum der Universität Würzburg** (Dr. Irma Wehgartner)
  - Amphora L 222 (multiple photographs, Figure 8.10 on page 124 and Figure 8.19 on page 130)
- **Universitätsbibliothek Würzburg** (Dr. Hans-Günter Schmidt)
  - Animation Fries-Chronik UB Würzburg, M.ch.f.760, fol. 1r (Figure 8.20 on page 130)
- **NASA (National Aeronautics and Space Administration)**
  - Blue Marble Next Generation, January (Top image of Plate I, page 173, taken from the *Visible Earth* website [77])