БЪЛГАРСКА АКАДЕМИЯ НА НАУКИТЕ
ИНСТИТУТ ПО ИНФОРМАЦИОННИ И КОМУНИКАЦИОННИ ТЕХНОЛОГИИ

Stoyan Milkov Mihov

Finite-State Automata, Transducers and Bimachines: Algorithmic Constructions and Implementations

ABSTRACT

OF DISSERTATION
for awarding of
the scientific degree "Doctor of Science"
in the professional field 4.6.
Informatics and Computer science

Sofia, 2019

The dissertation was discussed and admitted for defence during an extended meeting of the Linguistic Modeling and Knowledge Processing Department of IICT-BAS, held on 05.12.2019.

The dissertation contains 226 pages, 8 chapters, preface, conclusion and bibliography, 38 figures, and 5 pages of references, which include 48 titles.

The dissertation defense will be held on the ..... at ..... in room 218 of block 25A of IICT-BAS at an open meeting of the scientific jury with the following members:

1. Prof. D.Sc. Galia Angelova
2. Corr.Memb. D.Sc. Svetozar Margenov
3. Prof. D.Sc. Ivan Dimov
4. Acad. D.Sc. Veselin Drenski
5. Prof. D.Sc. Ivan Landzhev
6. Prof. Dr. Tinko Tinchev
7. Assoc. Prof. Dr. Hristo Ganchev

The materials for the defence are available to those interested in Room 215 of IICT-BAS, Acad. G. Bonchev Str., bl. 25A.
Author: Stoyan Milkov Mihov

Title: FINITE-STATE AUTOMATA, TRANSDUCERS AND BIMACHINES: ALGORITHMIC CONSTRUCTIONS AND IMPLEMENTATIONS

General Characteristics of the Dissertation

Interest in the Topic and Overview of the Main Results in the Field

Finite-state techniques provide theoretically elegant and computationally efficient solutions for various (hard, non-trivial) problems in text and natural language processing [Roche and Schabes, 1997b, Mohri, 1997, Karttunen et al., 1997a], speech processing [Mohri et al., 2008], pattern matching [Navarro and Raffinot, 2002], knowledge representation [Angelova and Mihov, 2008], and many others. Due to its importance in many fundamental applications, the theory of finite-state automata and related finite-state machines has been studied intensively and its development still continues.

The theory of finite-state automata has been described from a computational point of view in numerous textbooks (e.g. [Hopcroft et al., 2006, Kozen, 1997, Lewis and Papadimitriou, 1998]). These books are mainly about finite-state automata and regular languages over a free monoid and present mostly the basic automata properties such as: the Kleene theorem for the equivalence of regular languages with finite-state automata languages, the determinization of finite-state automata, the closures with respect to intersection and complement, the Myhill-Nerod relation and the minimization of finite-state automata, and constructions of finite-state automata from regular expressions.

From a theoretical-algebraic point of view, finite automata have been studied in the monographs [Eilenberg, 1974, Eilenberg, 1976, Berstel, 1979, Sakarovitch, 2009]. In these books, in addition to the classical case of free monoids, finite state automata over arbitrary monoids are also considered. These works also explore a number of additional algebraic properties, as well as properties of finite-state transducers.

A presentation focused on the applications of the finite-state machines for searching and natural language processing is presented in the works [Kaplan and Kay, 1994, Mohri, 1996, Roche and Schabes, 1997b, Navarro and Raffinot, 2002, Beesley and Karttunen, 2003, Maurel and Guenthner, 2005]. One of the most common applications is the use of finite-state transducers for application and representation of replace rules. These techniques are presented in, e.g. [Mohri and Sproat, 1996, Karttunen, 1997, Kaplan and Kay, 1994, Gerdemann and van Noord, 1999, Hulden, 2009].

Finite-state transducers and more specifically subsequential finite-state

transducers with outputs in numerical monoids are at the heart of modern speech recognition applications. They are described, for example, in [Mohri, 1997, Mohri et al., 2008]. The results for subsequential finite-state transducers are generalized for the case of other output monoids in [Gerdjikov and Mihov, 2017b, Gerdjikov and Mihov, 2017a].

Similarity search is another important application area of finite-state automata and transducers. Application of finite state techniques for text correction is described in [Ringlstetter et al., 2007, Mitankin et al., 2014]. In [Schulz and Mihov, 2002, Mihov and Schulz, 2004, Mitankin et al., 2011] efficient methodologies for approximate dictionary search and constructions of deterministic Levenshtein finite-state automata are presented.

Due to the complexity of their construction, the theory of bimachines is relatively poorly developed. After being introduced and studied in [Schützenberger, 1961, Reutenauer and Schützenberger, 1991], they are applied for natural language processing, for example, in [Roche and Schabes, 1997b]. In order to overcome the complexities of the bimachine constructions, in [Gerdjikov et al., 2017] we introduce a new bimachine construction, which avoids the preliminary step for constructing an intermedate unambiguous transducer. For certain classes of transducers this construction is shown to lead to exponentially smaller number of states in the resulting bimachine.

There are numerous implementations of software libraries for the construction and application of finite-state automata and tansducers. The most widely used systems that also offer transducer constructions are [Karttunen et al., 1997b, Schmid, 2006, Allauzen et al., 2007].

Aims and Objectives of the Dissertation

The aim of the dissertation is to describe the recent advances of the finite state technology, following a combined mathematical and implementational point of view. Though concepts are introduced in a mathematically rigorous way and correctness proofs for all procedures are given, the dissertation is not meant as a purely theoretical presentation of the subject. The goal of the dissertation is to provide both – formal construction for finite-state machines with correctness proofs and working implementations of all presented constructions together with corresponding documentation. These allows one to understand and implement complex finite-state based procedures for practically relevant tasks.

Methodology

The presentation in the dissertation follows the following principles:

1. *Theoretical generalisations aiming the extension of the scope of applicability.*
   The spectrum of finite-state machines that are covered is not restricted to classical finite-state automata and "recognition" tools. We also treat important "input-output" and "translation" devices such as multi-tape automata, finite-state transducers and bimachines. All these machines can be used, e.g., for efficient text rewriting, information extraction from textual corpora, and morphological analysis.

2. *Practical feasibility of the created abstract constructions.*
   After a conceptual introduction, full implementations/executable programs are given for all procedures, including a documentation of the programming code. In this way it is possible to observe the concrete behaviour of algorithms for arbitrary examples. It is also not difficult to enhance the given programs by means of self-written trace functionalities, which helps to even more thorough exploration of particular details of the algorithms and programs presented.

3. *Applicability to substantial practical problems.*
   Conceptual descriptions and implementations in a natural way lead to an intermediate goal reached at the end of the dissertation. Here we show also how to use the technology introduced for solving some important practical problems like spelling correction, phonetization, bignum arithmetics and others.

Approbation of the Results

Results included in the dissertation are presented at:

1. Conference on Mathematical Logic, SU "St. Kliment Ohridski", Gyolechitsa, 12 May 2018 .

2. Conference on Mathematical Logic, SU "St. Kliment Ohridski", Gyolechitsa, 7 October 2016 .

3. Gastvortrag, Centrum für Informations und Sprachverarbeitung (CIS), Ludwig-Maximilians Universität München, 23 Februar 2015

4. Conference on Mathematical Logic, SU "St. Kliment Ohridski", Gyolechitsa, 19 September 2014 .

Some of the results were reported at the following international conferences:

1. 11th International Conference on Language and Automata Theory and Applications, LATA 2017; Umea; Sweden; 6 March 2017

2. 22nd International Conference on Implementation and Application of Automata, CIAA 2017; Marne-la-Vallee; France; 27 June 2017

3. International Conference on Advanced Computing for Innovation, AComIn 2015; Sofia; Bulgaria; 10 November 2015

4. 1st International Conference on Digital Access to Textual Cultural Heritage, DATeCH 2014; Madrid; Spain; 19 May 2014

5. 11th IAPR International Workshop on Document Analysis Systems, DAS 2014; Tours; France; 7 April 2014

6. 12th International Conference on Document Analysis and Recognition, ICDAR 2013; Washington, DC; United States; 25 August 2013

7. 7th Conference on Computability in Europe, CiE 2011; Sofia; Bulgaria; 27 June 2011

Dissertation Publications

The presented dissertation essentially covers Chapters 1-8 of the monograph:

- Mihov, S. and Schulz, K. (2019). *Finite-State Techniques: Automata, Transducers and Bimachines*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press.

Results presented in the dissertation are published in 11 articles and 1 book chapter – 3 of the articles are published in journals with IMPACT factor and 7 in journals and proceedings with SJR factor. There are 227 citations (without self-citations) of those papers registered in SCOPUS.

1. Angelova, G. and Mihov, S. (2008). Finite state automata and simple conceptual graphs with binary conceptual relations. In *Supplementary Proceedings of the 16th International Conference on Conceptual Structures, ICCS 2008, Toulouse, France, July 7-11, 2008*, pages 139–148.

2. Daciuk, J., Mihov, S., Watson, B., and Watson, R. (2000). Incremental construction of minimal acyclic finite state automata. *Computational Linguistics*, 26(1):3–16.
IMPACT factor: Q1, SCOPUS citations: 84

3. Ganchev, H., Mihov, S., and Schulz, K. U. (2008). One-letter automata: How to reduce k tapes to one. In Hamm, F and Kepser, S, editor, *Logics For Linguistic Structures*, volume 201 of *Trends in Linguistics-Studies and Monographs*, pages 35–55.

4. Gerdjikov, S. and Mihov, S. (2017a). Myhill-nerode relation for sequentiable structures. *CoRR*, abs/1706.02910.

5. Gerdjikov, S. and Mihov, S. (2017b). Over which monoids is the transducer determinization procedure applicable? In *Language and Automata Theory and Applications - 11th International Conference, LATA 2017, Umeå, Sweden, March 6-9, 2017, Proceedings*, volume 10168 LNCS, pages 380–392.

6. Gerdjikov, S., Mihov, S., and Schulz, K. U. (2017). A simple method for building bimachines from functional finite-state transducers. In Carayol, A. and Nicaud, C., editors, *Implementation and Application of Automata*, volume 10329 LNCS, pages 113–125. Springer International Publishing.

7. Mihov, S. and Maurel, D. (2001). Direct construction of minimal acyclic subsequential transducers. In *Proceedings of the Conference on Implementation and Application of Automata CIAA'2000*, volume 2088 of *LNCS*, pages 217–229. Springer.
SCOPUS citations: 3

8. Mihov, S. and Schulz, K. U. (2004). Fast approximate search in large dictionaries. *Computational Linguistics*, 30(4):451–477.
IMPACT factor: Q1, SCOPUS citations: 45

9. Mitankin, P., Gerdjikov, S., and Mihov, S. (2014). An approach to unsupervised historical text normalisation. In *Digital Access to Textual Cultural Heritage 2014, DATeCH 2014, Madrid, Spain, May 19-20, 2014*, pages 29–34.
SCOPUS citations: 3

10. Mitankin, P., Mihov, S., and Schulz, K. U. (2011). Deciding word neighborhood with universal neighborhood automata. *Theoretical*

Computer Science, 412(22):2340–2355.
IMPACT factor: Q3, SCOPUS citations: 1

11. Ringlstetter, C., Schulz, K. U., and Mihov, S. (2007). Adaptive text correction with web-crawled domain-dependent dictionaries. *ACM Transactions on Speech and Language Processing*, 4(4).
SCOPUS citations: 10

12. Schulz, K. U. and Mihov, S. (2002). Fast String Correction with Levenshtein-Automata. *International Journal of Document Analysis and Recognition*, 5(1):67–85.
SCOPUS citations: 81

Content of the Dissertation

The dissertation contains 226 pages, 8 chapters, preface, conclusion and bibliography, 38 figures, and 5 pages of references, which include 48 titles. The main body of the dissertation covers Chapters 1-8 of the monograph [Mihov and Schulz, 2019].

1 Formal preliminaries

The aim of this chapter is twofold. First, we recall a collection of basic mathematical notions that are needed for the discussions of the following chapters. Second, we have a first - still purely mathematical - look at the central topics of the dissertation: languages, relations and functions between strings, as well as important operations on languages, relations and functions. We also introduce monoids, a class of algebraic structures that gives an abstract view on strings, languages, and relations.

1.1 Sets, functions and relations

In this section the definitions and the notations for sets, n-tuples, relations and functions are introduced. We define the operations composition of relations and functions, projection and inversion of relations, image of a set by a binary relations, as well as the reflexive and transitive closure of binary relations (Kleene relational star R^*). For binary relations, the concepts of reflexivity, symmetry, transitivity and antisymmetry are introduced. The concepts of equivalence relation and equivalence classes, refinement of a equivalence

relation, as well as injectivity, surjectivity and bijectivity of functions are defined in the standard way.

Convension: In what follows, by a function we always mean a partial function if not mentioned otherwise. As a general convention, when writing an expression $f(m)$ we always mean that f is defined for m. An expression $!f(n)$ means that f is defined for n.

The following concepts are more specific:

Definition 1.1.6 A binary relation $R \subseteq M \times N$ is *infinitely ambiguous* iff there exists an $m \in M$ such that the set $R(\{m\})$ is infinite.

Definition 1.1.9 Let $R \subseteq \prod_{i=1}^{n} M_i$ where $n \geq 2$. The relation

$$R_{\times i} := \{\bar{m}_{\times i} \mid R(\bar{m})\}$$

is called the *projection of R w.r.t. the set of coordinates* $\{1, \ldots, i-1, i+1, \ldots, n\}$. Let $\emptyset \neq \{i_1, \ldots, i_k\} \subseteq \{1, \ldots, n\}$. Then

$$Proj(\langle i_1, \ldots, i_k \rangle, R) := \{\langle m_{i_1}, \ldots, m_{i_k} \rangle \mid \langle m_1, \ldots, m_n \rangle \in R\}$$

is called the *generalized projection of R w.r.t. the sequence of coordinates* $\langle i_1, \ldots, i_k \rangle$.

Definition 1.1.16 Let $R \subseteq \prod_{i=1}^{n} M_i$ where $n \geq 2$. Let $I = \langle i_1, \ldots, i_k \rangle$ and $J = \langle j_1, \ldots, j_l \rangle$ be two sequences where $\{i_1, \ldots, i_k\}$ and $\{j_1, \ldots, j_l\}$ are non-empty subsets of the index set $\{1, \ldots, n\}$. Then $Func(I, J, R)$ denotes the function that maps each element $\langle m_{i_1}, \ldots, m_{i_k} \rangle$ of $Proj(I, R)$ to the set

$$\{\langle m_{j_1}, \ldots, m_{j_l} \rangle \in Proj(J, R) \mid \exists \langle m_1, \ldots m_n \rangle \in R\}.$$

1.2 Lifting functions to sets and tuples

We now introduce two ways of "lifting" a function that are used later at many places.

Definition 1.2.1 Let $f : M \to N$ be a function. The "set-lifted" version of f is the function $\hat{f} : 2^M \to 2^N$ defined pointwise:

$$\hat{f}(T) = \{f(t) \,|\, t \in T \cap dom(f)\}.$$

The function $\hat{f}$ is a total function – $\hat{f}(T)$ is defined for any $T \subseteq M$.

Later we will simply use the symbol f to denote a lifted version if this does not lead to confusion.

Proposition 1.2.3 *Let $f : M \to N$ be a function, let $T_i \subseteq M$ for every $i \in I$. Then*

$$f(\bigcup_{i \in I} T_i) = \bigcup_{i \in I} f(T_i).$$

Definition 1.2.4 Let $f : M^k \to N$ be a k-ary function, let $n \geq 2$. The "tuple-lifted" version of f is the k-ary function $\bar{f} : (M^n)^k \to N^n$ defined component-wise:

$$\begin{aligned} & \bar{f}(\langle m_{1,1}, \ldots, m_{1,n} \rangle, \ldots, \langle m_{k,1}, \ldots, m_{k,n} \rangle) \\ = \; & \langle f(m_{1,1}, \ldots, m_{k,1}), \ldots, f(m_{1,n}, \ldots, m_{k,n}) \rangle. \end{aligned}$$

1.3 Alphabets, words and languages

Words, languages, and operations on languages are the formal basis of classic finite state machines. In this section, these concepts are introduced in the standard way.

An *alphabet* Σ is a set of symbols. If not mentioned otherwise, alphabets are assumed to be non-empty and finite.

Definition 1.3.1 Let Σ be an alphabet. A *word* w over Σ is an n-tuple

$$w = \langle a_1, \ldots, a_n \rangle$$

where $n \geq 0$ and $a_i \in \Sigma$ for $i = 1, \ldots, n$. The integer n is called the *length* of w and denoted $|w|$. The empty tuple $\langle\rangle = \varepsilon$, which has length 0, is called the *empty word*. Σ^* denotes the set of all words over Σ, and $\Sigma^\varepsilon := \Sigma \cup \{\varepsilon\}$. The *concatenation* of two words $u = \langle a_1, \ldots, a_n \rangle$ and $v = \langle b_1, \ldots, b_m \rangle \in \Sigma^*$ is

$$u \cdot v := \langle a_1, \ldots, a_n, b_1, \ldots, b_m \rangle .$$

Definition 1.3.3 Let $t \in \Sigma^*$, assume that t can be represented in the form $t = u \cdot v \cdot w$ for some $u, v, w \in \Sigma^*$. Then $v \in \Sigma^*$ is an *infix* of t. If $u = \varepsilon$, then v is a *prefix* of t. If $w = \varepsilon$, then v is a *suffix* of t. The notation $v \leq t$ ($v < t$) expresses that v is a (proper) prefix of t.

Definition 1.3.9 Let Σ be an alphabet. A subset $L \subseteq \Sigma^*$ is called a *language* over Σ.

Definition 1.3.10 Let L_1, L_2 be classical languages over the alphabet Σ. Then

$$L_1 \cdot L_2 := \{w_1 \cdot w_2 \mid w_1 \in L_1, w_2 \in L_2\}$$

is called the *concatenation* of L_1 and L_2.

Definition 1.3.13 Let L be a language. We inductively define

1. $L^0 = \{\varepsilon\}$,
2. $L^{k+1} = L^k \cdot L$,

The language $L^* = \bigcup_{k=0}^{\infty} L^k$ is called the *Kleene star* of L.

1.4 Word tuples, string relations and string functions

For all notions introduced in the previous section there are natural generalizations when moving from elements to n-tuples.

Definition 1.4.1 The *concatenation of n-tuples of words* (or *n-way concatenation of words*) is defined as

$$\langle u_1, \ldots, u_n \rangle \mathbin{\bar{\cdot}} \langle v_1, \ldots, v_n \rangle := \langle u_1 \cdot v_1, \ldots, u_n \cdot v_n \rangle.$$

Definition 1.4.6 Let $n \geq 1$. For each $i = 1, \ldots, n$, let Σ_i be an alphabet. Then each set $R \subseteq \prod_{i=1}^{n} \Sigma_i^*$ is called an *n-ary string relation*.

Definition 1.4.8 Let R_1, R_2 be n-ary string relations. The *concatenation* of R_1 and R_2 is

$$R_1 \mathbin{\bar{\cdot}} R_2 := \{\bar{u} \mathbin{\bar{\cdot}} \bar{v} \mid \bar{u} \in R_1, \bar{v} \in R_2\}.$$

Definition 1.4.12 The *(concatenation) Kleene star* for an n-ary string relation R is defined as $R^* = \bigcup_{k=0}^{\infty} R^k$, where

- $R^0 = \{\bar{\varepsilon}\}$,
- $R^{k+1} = R^k \mathbin{\bar{\cdot}} R$,

1.5 The general monoidal perspective

In the last two sections we introduced algebraic structures that describe strings and n-tuples of strings. These two structures are special cases of the general concept of a monoid. In this section we introduce the general concept of a monoid, and develop a kind of rudimentary "formal language theory" for general monoids.

Definition 1.5.1 A *monoid* $\mathcal{M}$ is a triple $\langle M, \circ, e \rangle$ where

- M is a non-empty set, the set of monoid elements,

- $\circ : M \times M \to M$ is the monoid operation (we will use infix notation),
- $e \in M$ is the monoid unit element,

and the following conditions hold:

- $\forall a, b, c \in M : a \circ (b \circ c) = (a \circ b) \circ c$ (associativity of "$\circ$"),
- $\forall a \in M : a \circ e = e \circ a = a$ (e is a unit element).

Definition 1.5.4 Let $\mathcal{M} = \langle M, \circ, e \rangle$ be a monoid. A *monoidal language over* $\mathcal{M}$ is a subset of M.

Definition 1.5.5 Let $\mathcal{M} = \langle M, \circ, e \rangle$ be a monoid. For $T_1, T_2 \subseteq M$ the set

$$T_1 \circ T_2 := \{t_1 \circ t_2 \,|\, t_1 \in T_1, t_2 \in T_2\}.$$

is called the *monoidal product* of the monoidal languages T_1 and T_2.

Definition 1.5.6 Let $\mathcal{M} = \langle M, \circ, e \rangle$ be a monoid and $T \subseteq M$. We inductively define

1. $T^0 = \{e\}$,
2. $T^{k+1} = T^k \circ T$,

We call $T^* = \bigcup_{k=0}^{\infty} T^k$ the *iteration* or (monoidal) *Kleene star* of T.

Definition 1.5.7 A subset $T \subseteq M$ of a monoid $\mathcal{M} = \langle M, \circ, e \rangle$ is a *submonoid* of $\mathcal{M}$ iff $e \in T$ and $T^2 \subseteq T$.

Proposition 1.5.8 *For any subset $T \subseteq M$ of a monoid $\mathcal{M}$, the set T^* is the smallest submonoid of $\mathcal{M}$ containing T.*

Definition 1.5.9 Let $\mathcal{M}_1 = \langle M_1, \circ, e_1 \rangle$ and $\mathcal{M}_2 = \langle M_2, \bullet, e_2 \rangle$ be monoids. A total function $h : M_1 \to M_2$ is a *monoid homomorphism* iff the following conditions hold:

- $h(e_1) = e_2$,
- $\forall a, b \in M_1 : h(a \circ b) = h(a) \bullet h(b)$.

Proposition 1.5.14 *Let $\mathcal{M} = \langle M, \circ, e \rangle$ be a monoid, let Σ be an alphabet and $f : \Sigma \to M$ be a total function. Then the* natural extension h_f *of f over Σ^*, inductively defined as*

1. $h_f(\varepsilon) = e$

2. $h_f(\alpha \cdot a) = h_f(\alpha) \circ f(a)$, *where* $\alpha \in \Sigma^*, a \in \Sigma$,

is a homomorphism between the monoids Σ^* *and* $\mathcal{M}$ *and the unique homomorphism extending* f.

Definition 1.5.15 Let $n \geq 1$, for $1 \leq i \leq n$ let $\mathcal{M}_i = \langle M_i, \circ_i, e_i \rangle$ be a monoid. Let $\bar{e} := \langle e_1, \ldots, e_n \rangle$ and let $\bar{\circ} : (\prod_{i=1}^n M_i) \times (\prod_{i=1}^n M_i) \to \prod_{i=1}^n M_i$ denote the function

$$\langle u_1, \ldots, u_n \rangle \,\bar{\circ}\, \langle v_1, \ldots, v_n \rangle := \langle u_1 \circ_1 v_1, \ldots, u_n \circ_n v_n \rangle .$$

Then the triple $\prod_{i=1}^n \mathcal{M}_i := \langle \prod_{i=1}^n M_i, \bar{\circ}, \bar{e} \rangle$ is called the *Cartesian product* of the monoids $\mathcal{M}_i$.

2 Monoidal finite-state automata

Here we immediately look at the more general concept of a *monoidal* finite-state automaton, and the focus of this chapter are *general* constructions and results for finite-state automata over *arbitrary monoids* and monoidal languages.

2.1 Basic concept and examples

We introduce the central concept of this chapter.

Definition 2.1.1 A *monoidal finite-state automaton* (MSA) is a tuple of the form $\mathcal{A} = \langle \mathcal{M}, Q, I, F, \Delta \rangle$ where

- $\mathcal{M} = \langle M, \circ, e \rangle$ is a monoid,
- Q is a finite *set of states*,
- $I \subseteq Q$ is the set of *initial states*,
- $F \subseteq Q$ is the set of *final states*, and
- $\Delta \subseteq Q \times M \times Q$ is a finite set called the *transition relation*.

Triples $\langle p, m, q \rangle \in \Delta$ are called *transitions*. The transition $\langle p, m, q \rangle$ *begins* at p, *ends* at q and has the *label* m.

Definition 2.1.3 *Classical finite-state automata* are monoidal finite-state automata where the underlying monoid is the free monoid over a finite alphabet Σ and the transition labels are in $\Sigma^\varepsilon = \Sigma \cup \{\varepsilon\}$.

Definition 2.1.7 Let $\mathcal{A} = \langle \mathcal{M}, Q, I, F, \Delta \rangle$ be a monoidal finite-state automaton. A *proper path* in $\mathcal{A}$ is a finite sequence of $k > 0$ transitions

$$\pi = \langle q_0, a_1, q_1 \rangle \langle q_1, a_2, q_2 \rangle \dots \langle q_{k-1}, a_k, q_k \rangle$$

where $\langle q_{i-1}, a_i, q_i \rangle \in \Delta$ for $i = 1 \dots k$. The number k is called the *length* of π, we say that π starts in q_0 and ends in q_k. States $q_0, \dots, q_k$ are *the states on the path* π. The monoid element $w = a_1 \circ \dots \circ a_k$ is called the *label* of π. We may denote the path π as

$$\pi = q_0 \to^{a_1} q_1 \dots \to^{a_k} q_k.$$

The *null path* of $q \in Q$ is 0_q starting and ending in q with label e. A *successful path* is a path starting in an initial state and ending in a final state.

Definition 2.1.10 Let $\mathcal{A}$ be as above. Then the set of all labels of successful paths of $\mathcal{A}$ is called the *monoidal language accepted* (or *recognized*) by $\mathcal{A}$ and is denoted $L(\mathcal{A})$.

Definition 2.1.14 Let $\mathcal{A}$ be a monoidal finite-state automata. The *generalized transition relation* Δ^* is defined as the smallest subset of $Q \times M \times Q$ with the following closure properties:

- for all $q \in Q$ we have $\langle q, e, q \rangle \in \Delta^*$.
- For all $q_1, q_2, q_3 \in Q$ and $w, a \in M$: if $\langle q_1, w, q_2 \rangle \in \Delta^*$ and $\langle q_2, a, q_3 \rangle \in \Delta$, then also $\langle q_1, w \circ a, q_3 \rangle \in \Delta^*$.

Theorem 2.1.22 *Any monoidal finite-state automaton is the homomorphic image of a classical finite-state automaton. Any monoidal automaton language can be obtained as a homomorphic image of a classical automaton language.*

2.2 Closure properties of monoidal finite-state automata

We show that the class of monoidal automaton languages is closed under the main monoidal operations.

Proposition 2.2.1 *The class of monoidal automaton languages is closed under monoid homomorphisms. The class of monoidal automaton languages over a given monoid $\mathcal{M}$ is closed under the regular operations union, monoidal product, and monoidal Kleene-Star.*

2.3 Monoidal regular languages and monoidal regular expressions

Definition 2.3.1 Let $\mathcal{M} = \langle M, \circ, e \rangle$ be a monoid. We define the class of *monoidal regular languages* over $\mathcal{M}$ by induction:

1. $\emptyset$ is a monoidal regular language over $\mathcal{M}$;
2. if $m \in M$, then $\{m\}$ is a monoidal regular language over $\mathcal{M}$;
3. if $L_1, L_2 \subseteq M$ are monoidal regular languages over $\mathcal{M}$, then
   - $L_1 \cup L_2$ is a monoidal regular language over $\mathcal{M}$ (union),
   - $L_1 \circ L_2$ is a monoidal language over $\mathcal{M}$ (monoidal product, cf. Def. 1.5.5),
   - L_1^* is a monoidal regular language over $\mathcal{M}$ (monoidal Kleene star, cf. Def. 1.5.6).

Definition 2.3.5 Let $\mathcal{M} = \langle M, \circ, e \rangle$ be a monoid. A *monoidal regular expression* over $\mathcal{M}$ for $M \cap \{(,), *, +, \cdot, \emptyset\} = \emptyset$ is a word over $M \cup \{(,), *, +, \cdot, \emptyset\}$.The set of monoidal regular expressions over $\mathcal{M}$ is defined by induction:

1. $\emptyset$ is a monoidal regular expression over $\mathcal{M}$;
2. if $m \in M$, then m is a monoidal regular expression over $\mathcal{M}$;
3. if E_1 and E_2 are monoidal regular expressions over $\mathcal{M}$, then
   - $(E_1 + E_2)$ is a monoidal regular expression over $\mathcal{M}$,
   - $(E_1 \cdot E_2)$ is a monoidal regular expression over $\mathcal{M}$,
   - (E_1^*) is a monoidal regular expression over $\mathcal{M}$.

2.4 Equivalence between monoidal regular languages and monoidal automaton languages

In this section we show that monoidal regular expressions and monoidal finite-state automata yield two descriptions of the same class of languages.

Proposition 2.4.1

1. *(Empty language) For* $\mathcal{A}_\emptyset = \langle \mathcal{M}, \emptyset, \emptyset, \emptyset, \emptyset \rangle$ *we have* $L(\mathcal{A}_\emptyset) = \emptyset$.

2. *(Single element languages) Let $m \in M$. For the monoidal finite-state automaton $\mathcal{A}_m = \langle \mathcal{M}, \{q_0, q_1\}, \{q_0\}, \{q_1\}, \{\langle q_0, m, q_1 \rangle\} \rangle$ we have $L(\mathcal{A}_m) = \{m\}$.*

Theorem 2.4.2 (Kleene) *A monoidal language is regular if and only if it is a monoidal automaton language.*

2.5 Simplifying the structure of monoidal finite-state automata

In contrast to the operations considered in Section 2.2 the operations considered here do not modify the monoidal language of the given automaton. The goal is rather to simplify automata from a structural point of view, thus facilitating the recognition process.

Definition 2.5.1 A monoidal finite-state automaton $\mathcal{A} = \langle \mathcal{M}, Q, I, F, \Delta \rangle$ is *trimmed* iff each state $q \in Q$ is on a successful path of $\mathcal{A}$.

It is straightforward to check for a given state $q \in Q$ of a given monoidal finite-state automaton $\mathcal{A}$ if it is on a successful path. In the negative case we may delete q and all transitions leading to or departing from q. In this way, a trimmed monoidal finite-state automaton $\mathcal{A}'$ equivalent to $\mathcal{A}$ is obtained.

Definition 2.5.2 Let $\mathcal{A} = \langle \mathcal{M}, Q, I, F, \Delta \rangle$ be a monoidal finite-state automaton where $\mathcal{M} = \langle M, \circ, e \rangle$. $\mathcal{A}$ is called *e-free* iff $\Delta \subseteq Q \times (M \setminus \{e\}) \times Q$.

Proposition 2.5.4 *For any monoidal finite-state automaton $\mathcal{A} = \langle \mathcal{M}, Q, I, F, \Delta \rangle$ there exists an equivalent e-free monoidal finite-state automaton $\mathcal{A}$' with the same set of states.*

Proposition 2.5.6 *For any monoidal finite-state automaton $\mathcal{A} = \langle \mathcal{M}, Q, I, F, \Delta \rangle$ there exists an equivalent e-free monoidal finite-state automaton $\mathcal{A}$' with the same set of states such that for each $q \in Q$ we have $L_{\mathcal{A}}(q) = L_{\mathcal{A}'}(q)$.*

3 Classical finite-state automata and regular languages

Classical finite-state automata represent the most important class of monoidal finite-state automata. Since the underlying monoid is free, this class of automata has several interesting specific features. In this chapter we present the main properties in regards to determinization and minimization of classical finite-state automata.

3.1 Deterministic finite-state automata

Definition 3.1.1 Let $\mathcal{A} = \langle \Sigma, Q, I, F, \Delta \rangle$ be a classical finite-state automaton. $\mathcal{A}$ is *deterministic* iff the following conditions hold:

- $\mathcal{A}$ has exactly one initial state q_0,
- all transition labels are letters in Σ and
- $\langle p, a, q \rangle \in \Delta$ and $\langle p, a, q' \rangle \in \Delta$ implies $q = q'$ for all triples in Δ.

In this situation the transition relation can be described as a (partial) function $\delta : Q \times \Sigma \to Q$.

Deterministic finite-state automata are often represented in the form

$$\langle \Sigma, Q, q_0, F, \delta \rangle$$

.

Definition 3.1.4 Let $D \subseteq \Sigma^*$ be a finite set of words over the alphabet Σ. The *trie* for D is the deterministic finite-state automaton

$$\mathcal{A}_D = \langle \Sigma, \mathit{Pref}(D), \varepsilon, D, \delta \rangle$$

where

$$\delta = \{ \langle \alpha, \sigma, \alpha \cdot \sigma \rangle \mid \sigma \in \Sigma \ \& \ \alpha, \alpha \cdot \sigma \in \mathit{Pref}(D) \}.$$

The proof of the following proposition is obvious.

Proposition 3.1.5 *Let $D \subseteq \Sigma^*$ be a finite set of words. Then*

1. *$L(\mathcal{A}_D) = D$,*
2. *Each state q of $\mathcal{A}_D$ can be reached on exactly one path from the initial state ε. The label of this path is the string q.*

3.2 Determinization of classical finite-state automata

Our next aim is to show that each classical finite-state automaton can be effectively converted to an equivalent deterministic classical finite-state automaton.

Proposition 3.2.1 *Every monoidal finite-state automaton $\mathcal{A} = \langle \Sigma, Q, I, F, \Delta \rangle$ over the free monoid Σ^* can be converted to an equivalent classical finite-state automaton $\mathcal{A}'$ without ε-transitions with a set of states Q' such that $Q \subseteq Q'$ and for each $q \in Q$ we have $L_{\mathcal{A}}(q) = L_{\mathcal{A}'}(q)$.*

Theorem 3.2.2 (Determinization of classical finite-state automata)

Every classical finite-state automaton $\mathcal{A} = \langle \Sigma, Q, I, F, \Delta \rangle$ can be converted to an equivalent deterministic finite-state automaton $\mathcal{A}_D$ with total transition function.

3.3 Additional closure properties for classical finite-state automata

Proposition 3.3.1 (Complementing deterministic FSA) *Let*

$$\mathcal{A} = \langle \Sigma, Q, q_0, F, \delta \rangle$$

be a deterministic finite-state automaton where δ is a total function. Let

$$\mathcal{A}' = \langle \Sigma, Q, q_0, Q \setminus F, \delta \rangle .$$

Then $L(\mathcal{A}') = \Sigma^* \setminus L(\mathcal{A})$.

Proposition 3.3.2 *Let $\mathcal{A}_1 = \langle \Sigma, Q_1, I_1, F_1, \Delta_1 \rangle$ and $\mathcal{A}_2 = \langle \Sigma, Q_2, I_2, F_2, \Delta_2 \rangle$ be two classical ε-free finite-state automata. Then the following holds:*

1. *(Intersection for ε-free classical finite-state automata) For the finite-state automaton*

   $$\mathcal{A} := \langle \Sigma, Q_1 \times Q_2, I_1 \times I_2, F_1 \times F_2, \Delta' \rangle$$

   where $\Delta' := \{ \langle \langle q_1, q_2 \rangle, a, \langle r_1, r_2 \rangle \rangle \mid \langle q_1, a, r_1 \rangle \in \Delta_1 \ \& \ \langle q_2, a, r_2 \rangle \in \Delta_2 \}$ we have $L(\mathcal{A}) = L(\mathcal{A}_1) \cap L(\mathcal{A}_2)$.

2. *(Difference for deterministic classical finite-state automata) If $\mathcal{A}_2$ is a deterministic classical finite-state automaton and the transition function of $\mathcal{A}_2$ is total, then for the finite-state automaton*

   $$\mathcal{A} := \langle \Sigma, Q_1 \times Q_2, I_1 \times I_2, F_1 \times (Q_2 \setminus F_2), \Delta' \rangle$$

   where $\Delta' := \{ \langle \langle q_1, q_2 \rangle, a, \langle r_1, r_2 \rangle \rangle \mid \langle q_1, a, r_1 \rangle \in \Delta_1 \ \& \ \langle q_2, a, r_2 \rangle \in \Delta_2 \}$ we have $L(\mathcal{A}) = L(\mathcal{A}_1) \setminus L(\mathcal{A}_2)$.

Proposition 3.3.3 *Let $\mathcal{A} := \langle \Sigma^*, Q, F, I, \Delta \rangle$ be a monoidal finite-state automaton over the free monoid Σ^*. Then for the monoidal finite-state automaton*

$$\mathcal{A}' := \langle \Sigma^*, Q, F, I, \Delta' \rangle$$

where $\Delta' = \{ \langle q_2, \rho(a), q_1 \rangle \mid \langle q_1, a, q_2 \rangle \in \Delta \}$ we have $L(\mathcal{A}') = \rho(L(\mathcal{A}))$.

Corollary 3.3.4 *The class of languages accepted by classical finite-state automata is closed under complement, intersection, set difference, and reversal.*

3.4 Minimal deterministic finite-state automata and the Myhill-Nerode equivalence relation

In order to find a minimal deterministic automaton for a classical language it is important to take an algebraic perspective.

Definition 3.4.1 An equivalence relation $R \subseteq \Sigma^* \times \Sigma^*$ is called *right invariant* if

$$\forall u, v \in \Sigma^* : u \; R \; v \rightarrow (\forall w \in \Sigma^* : u \cdot w \; R \; v \cdot w).$$

Definition 3.4.2 Let $L \subseteq \Sigma^*$ be a language, let $R \subseteq \Sigma^* \times \Sigma^*$ be an equivalence relation. L and R are called *compatible* if

$$\forall u, v \in \Sigma^* : (u \in L \;\&\; u \; R \; v) \rightarrow v \in L.$$

Proposition 3.4.4 *Let $R \subseteq \Sigma^* \times \Sigma^*$ be a right invariant equivalence relation such that the index of R is finite, let $L \subseteq \Sigma^*$ be a language over Σ compatible with R. Then for the deterministic classical finite-state automaton*

$$\mathcal{A}_{R,L} = \langle \Sigma, \{[s]_R \,|\, s \in \Sigma^*\}, [\varepsilon]_R, \{[s]_R \,|\, s \in L\}, \delta_R \rangle$$

with transition function $\delta_R = \{\langle [u]_R, a, [u \cdot a]_R \rangle \,|\, u \in \Sigma^*, a \in \Sigma\}$ we have $L(\mathcal{A}_{R,L}) = L$.

Proposition 3.4.7 *Let $\mathcal{A} = \langle \Sigma, Q, q_0, F, \delta \rangle$ be a deterministic classical finite-state automaton. Then*

1. *$R_{\mathcal{A}} := \{\langle u, v \rangle \in \Sigma^* \times \Sigma^* \,|\, \delta^*(q_0, u) = \delta^*(q_0, v)\}$ is a right invariant equivalence relation and $L(\mathcal{A})$ is compatible with $R_{\mathcal{A}}$,*
2. *the automaton $\mathcal{A}_{R_{\mathcal{A}}, L(\mathcal{A})}$ is isomorphic to $\mathcal{A}$ by the state renaming function $h : \{[s]_{R_{\mathcal{A}}} \,|\, s \in \Sigma^*\} \rightarrow Q$ defined as $h([w]_{R_{\mathcal{A}}}) = \delta^*(q_0, w)$.*

Definition 3.4.8 Let $L \subseteq \Sigma^*$ be a language over Σ. Then the relation

$$R_L = \{\langle u, v \rangle \in \Sigma^* \times \Sigma^* \,|\, \forall w \in \Sigma^* : u \cdot w \in L \text{ iff } v \cdot w \in L\}$$

is called the *Myhill-Nerode relation* for the language L.

Proposition 3.4.9 *Let $L \subseteq \Sigma^*$ be a language over Σ. Then the Myhill-Nerode relation R_L is a right invariant equivalence relation and R_L is compatible with L.*

Definition 3.4.10 Let R_L have finite index. Then the canonical deterministic automaton $\mathcal{A}_{R_L,L}$ for R_L and L is called the *Myhill-Nerode automaton* for the language L.

Proposition 3.4.12 *Let $\mathcal{A} = \langle \Sigma, Q, q_0, F, \delta \rangle$ be a classical deterministic finite-state automaton. Then $R_{\mathcal{A}} \subseteq R_{L(\mathcal{A})}$.*

Theorem 3.4.13 *For each classical deterministic finite-state automaton there exists a unique (up to renaming of states) equivalent deterministic finite-state automaton that is minimal with respect to the number of states.*

Theorem 3.4.14 *Let $L \subseteq \Sigma^*$ be a classical language. Then L is a classical automaton language iff the index of R_L is finite.*

Proposition 3.4.17 *A deterministic finite-state automaton $\mathcal{A} = (\Sigma, Q, q_0, F, \delta)$ is minimal iff distinct states of $\mathcal{A}$ are never equivalent.*

3.5 Minimization of deterministic finite-state automata

Given a deterministic finite-state automaton $\mathcal{A} = \langle \Sigma, Q, q_0, F, \delta \rangle$ for the language $L(\mathcal{A}) = L$ we now show how to build an equivalent minimal automaton $\mathcal{A}_L$ by simultaneously identifying all equivalent states.

Definition 3.5.1 The relations $R_i \subseteq Q \times Q$ $(i \geq 0)$ are formally defined as

$$q \; R_i \; p \;:\Leftrightarrow\; \forall \alpha \in \Sigma^* \text{ s.th. } |\alpha| \leq i : (\delta^*(q, \alpha) \in F \leftrightarrow \delta^*(p, \alpha) \in F).$$

Lemma 3.5.2 *For all states $q, p \in Q$ the following two conditions are equivalent:*

1. $\forall \alpha \in \Sigma^*$ *s.th.* $|\alpha| \leq i + 1 : (\delta^*(q, \alpha) \in F \leftrightarrow \delta^*(p, \alpha) \in F)$,
2. (a) $\forall \alpha \in \Sigma^*$ *s.th.* $|\alpha| \leq i : (\delta^*(q, \alpha) \in F \leftrightarrow \delta^*(p, \alpha) \in F)$, *and*
   (b) $\forall \sigma \in \Sigma, \forall \alpha \in \Sigma^*$ *s.th.* $|\alpha| \leq i : (\delta^*(\delta(q, \sigma), \alpha) \in F \leftrightarrow \delta^*(\delta(p, \sigma), \alpha) \in F)$.

Proposition 3.5.3

$$q \; R_{i+1} \; p \;\Leftrightarrow\; q \; R_i \; p \;\&\; \forall a \in \Sigma : \delta(q, a) \; R_i \; \delta(p, a)$$

Corollary 3.5.4 *Let $\mathcal{A} = \langle \Sigma, Q, q_0, F, \delta \rangle$ be a deterministic finite-state automaton where δ is total and each state is reachable. Then the relation $R = \bigcap_{i=0}^{\infty} R_i$, where*

$$\begin{aligned} q\ R_0\ p &\Leftrightarrow (q \in F \leftrightarrow p \in F) \\ q\ R_{i+1}\ p &\Leftrightarrow q\ R_i\ p\ \&\ \forall a \in \Sigma : \delta(q,a)\ R_i\ \delta(p,a) \end{aligned}$$

coincides with the equivalence of states $\equiv$ for $\mathcal{A}$. The automaton

$$\mathcal{A}' = \langle \Sigma, \{[q]_R \mid q \in Q\}, [q_0]_R, \{[f]_R \mid f \in F\}, \delta' \rangle$$

where $\delta'([q]_R), \sigma) := [\delta(q,\sigma)]_R$ is the minimal deterministic automaton equivalent to $\mathcal{A}$.

Definition 3.5.5 Let $g : Q \to X$. Then the equivalence relation $ker_Q(g) \subseteq Q \times Q$ defined as

$$\langle p, q \rangle \in ker_Q(g) \ :\Leftrightarrow\ g(p) = g(q)$$

is called the *kernel* of g over Q.

Proposition 3.5.6 *Let us define $f : Q \to \{0,1\}$*

$$f(q) := \begin{cases} 1 & \text{if } q \in F \\ 0 & \text{otherwise} \end{cases}$$

and for every $a \in \Sigma$ and $i \in \mathbb{N}$ the function $f_a^{(i)} : Q \to Q/R_i$ as

$$f_a^{(i)}(q) := [\delta(q,a)]_{R_i}.$$

Then

1. $R_0 = ker_Q(f)$
2. $R_{i+1} = \bigcap_{a \in \Sigma} ker_Q(f_a^{(i)}) \cap R_i$

Remark 3.5.9 For finite languages there exist direct methods for constructing the minimal deterministic finite-state automaton which provide better efficiency [Daciuk et al., 2000].

3.6 Coloured deterministic finite-state automata

We introduce a simple generalization of deterministic finite-states automata called coloured deterministic finite-state automata and show how this generalization can be represented by classical automata

Definition 3.6.1 Let C be a finite set called the *set of colours*. A *C-coloured deterministic finite-state automaton* is a deterministic finite-state automaton $\langle \Sigma, Q, q_0, F, \delta \rangle$ with total transition function δ together with a surjective total function $col : F \to C$. We write $\mathcal{A} = \langle \Sigma, C, Q, q_0, F, \delta, col \rangle$ for the coloured automaton. Colour $col(q)$ is called the colour of state $q \in F$. The *language $L(\mathcal{A})$ accepted by a coloured automaton $\mathcal{A}$* is defined as usual, ignoring colours.

Definition 3.6.2 Let $\mathcal{A} = \langle \Sigma, C, Q, q_0, F, \delta, col \rangle$ be a C-coloured deterministic finite-state automaton. The *colouring of words recognized by $\mathcal{A}$* is the function $col_{\mathcal{A}} : L(\mathcal{A}) \to C, w \mapsto col(\delta^*(q_0, w))$.

Proposition 3.6.3 *Let $L \subseteq \Sigma^*$ be a language and let $c : L \to C$ be a colouring function for the words of L. Consider the modified language*

$$L_c := \{\alpha \cdot c(\alpha) \mid \alpha \in L\} \subseteq \Sigma^* \cdot C.$$

Let $\mathcal{A} = \langle \Sigma, C, Q, q_0, F, \delta, col \rangle$ be a C-coloured deterministic finite-state automaton such that $L(\mathcal{A}) = L$ and $col_{\mathcal{A}} = c$. Let

$$\mathcal{A}_c := \langle \Sigma \cup C, Q \cup \{f\}, q_0, \{f\}, \delta \cup \{\langle q, col(q), f \rangle \mid q \in F\} \rangle$$

where $f \notin Q$ is a new state. Then $L(\mathcal{A}_c) = L_c$.

Using the correspondence from Proposition 3.6.3 it is possible to transfer classical automata results to the case of coloured automata, including minimization results.

Theorem 3.6.12 *For each coloured deterministic finite-state automaton there exists a unique (up to renaming of states) equivalent coloured deterministic finite-state automaton that is minimal with respect to the number of states.*

Proposition 3.6.14 *A coloured deterministic finite-state automaton $\mathcal{A}$ is minimal iff distinct states of $\mathcal{A}$ are never equivalent.*

Definition 3.6.15 We introduce equivalence relations R_i ($i \geq 0$) on Q, in this case inductively defining

$$\begin{aligned} q\ R_0\ p &\leftrightarrow (q \in F \leftrightarrow p \in F)\ \&\ (q \in F \to col(q) = col(p)) \\ q\ R_{i+1}\ p &\leftrightarrow q\ R_i\ p\ \&\ \forall a \in \Sigma : \delta(q, a)\ R_i\ \delta(p, a). \end{aligned}$$

Corollary 3.6.16 *Let $\mathcal{A} = \langle \Sigma, C, Q, q_0, F, \delta, col \rangle$ be a coloured deterministic finite-state automaton where δ is total and each state is reachable, let R_i defined as above ($i \geq 0$). Then $R = \bigcap_{i=0}^{\infty} R_i$ coincides with the equivalence of states $\equiv$ on $\mathcal{A}$. The coloured automaton*

$$\mathcal{A}' = \langle \Sigma, C, \{[q]_R \mid q \in Q\}, [q_0]_R, \{[f]_R \mid f \in F\}, \delta', col' \rangle$$

where $\delta'([q]_R), \sigma) := [\delta(q, \sigma)]_R$ and $col'([q]_R) := col(q)$ is the minimal coloured deterministic automaton equivalent to $\mathcal{A}$.

3.7 Pseudo-determinization and pseudo-minimization of monoidal finite-state automata

In this section we show that the definition of a deterministic monoidal finite-state automaton is not natural. can be represented as the homomorphic image of a classical finite-state automaton (cf. Theorem 2.1.22).

Definition 3.7.1 Let $\mathcal{M} = \langle M, \circ, e \rangle$ be a monoid. A monoidal finite-state automaton $\mathcal{A} = \langle \mathcal{M}, Q, I, F, \Delta \rangle$ is called *pseudo-deterministic* if there exists exactly one initial state in I and for any state $q \in Q$ and any $m \in M$ there exists at most one state q' such that $\langle q, m, q' \rangle \in \Delta$.

Proposition 3.7.2 *For each monoidal finite-state automaton $\mathcal{A}$ we may effectively construct a pseudo-deterministic monoidal finite-state automaton $\mathcal{A}'$ equivalent to $\mathcal{A}$.*

Definition 3.7.3 Let $\mathcal{M} = \langle M, \circ, e \rangle$ be a monoid. A pseudo-deterministic monoidal finite-state automaton $\mathcal{A} = \langle \mathcal{M}, Q, I, F, \Delta \rangle$ is called *pseudo-minimal* if the free companion of $\mathcal{A}$ is a minimal deterministic finite-state automaton.

Proposition 3.7.4 *For each monoidal finite-state automaton $\mathcal{A}$ we may effectively construct a pseudo-minimal monoidal finite-state automaton $\mathcal{A}'$ equivalent to $\mathcal{A}$.*

4 Monoidal multi-tape automata and finite-state transducers

An important generalization of classical finite-state automata are multi-tape automata, which are used for recognizing *relations* of a particular type. These relations offer a natural way to formalize all kinds of translations and transformations, which makes multi-tape automata interesting for many practical

applications and explains the general interest in this kind of device. From the perspective developed in the previous chapters, multi-tape automata represent a special subtype of monoidal automata. A natural subclass are monoidal finite-state transducers, which can be defined as two-tape automata where the first tape reads strings. In this chapter we present the most important properties of monoidal multi-tape automata in general and monoidal finite-state transducers in particular.

4.1 Monoidal multi-tape automata

We first introduce the general concept of a monoidal multi-tape automaton.

Definition 4.1.1 A *monoidal n-tape automaton* is a monoidal finite-state automaton $\mathcal{A} = \langle \prod_{i=1}^{n} \mathcal{M}_i, Q, I, F, \Delta \rangle$ over a monoid $\mathcal{M} = \mathcal{M}_1 \times \mathcal{M}_2 \ldots \times \mathcal{M}_n$ which is a Cartesian product of n monoids. If $n \geq 2$, the automaton is also called a *multi-tape automaton.*

In order to stress that the languages accepted by monoidal multi-tape automata are relations we introduce the following terminology.

Definition 4.1.5 A *monoidal n-tape automaton relation* is a monoidal language recognized by a monoidal n-tape automaton.

4.2 Additional closure properties of monoidal multi-tape automata

We now want to show that the class of relations accepted by monoidal multi-tape automata is closed under some additional operations. Some of the following constructions are obtained in a more elegant way if we add an e-loop to each state. We define

$$E(\Delta) := \Delta \cup \{\langle q, e, q \rangle \mid q \in Q\}.$$

Proposition 4.2.1

1. *(Cartesian product) Let* $\mathcal{A}_1 = \langle \mathcal{M}_1, Q_1, I_1, F_1, \Delta_1 \rangle$ *and* $\mathcal{A}_2 = \langle \mathcal{M}_2, Q_2, I_2, F_2, \Delta_2 \rangle$ *be two monoidal automata and let*

   $$\Delta := \{\langle \langle q_1, q_2 \rangle, \langle u_1, u_2 \rangle, \langle q_1', q_2' \rangle \rangle \mid \langle q_1, u_1, q_1' \rangle \in E(\Delta_1) \;\&\; \langle q_2, u_2, q_2' \rangle \in E(\Delta_2)\}.$$

   Then for the monoidal 2-tape automaton

   $$\mathcal{A} := \langle \mathcal{M}_1 \times \mathcal{M}_2, Q_1 \times Q_2, I_1 \times I_2, F_1 \times F_2, \Delta \rangle$$

   we have $L(\mathcal{A}) = L(\mathcal{A}_1) \times L(\mathcal{A}_2)$.

2. *(Projection) Let* $\mathcal{A} = \langle \mathcal{M}_1 \times \mathcal{M}_2 \times \ldots \times \mathcal{M}_n, Q, I, F, \Delta \rangle$ *be a monoidal* n*-tape automaton and* $n \geq 2$*. Let* $\Delta_{\times i} := \{\langle q, \bar{u}_{\times i}, q' \rangle \mid \langle q, \bar{u}, q' \rangle \in \Delta\}$*. Then for the monoidal* $(n-1)$*-tape automaton*

$$\mathcal{A}' := \langle \mathcal{M}_1 \times \ldots \times \mathcal{M}_{i-1} \times \mathcal{M}_{i+1} \times \ldots \times \mathcal{M}_n, Q, F, I, \Delta_{\times i} \rangle$$

we have $L(\mathcal{A}') = L(\mathcal{A})_{\times i}$*.*

3. *(Inverse relation for 2-tape automata) Let* $\mathcal{A} = \langle \mathcal{M}_1 \times \mathcal{M}_2, Q, I, F, \Delta \rangle$ *be a monoidal 2-tape automaton and*

$$\Delta' := \{\langle q_1, \langle v, u \rangle, q_2 \rangle \mid \langle q_1, \langle u, v \rangle, q_2 \rangle \in \Delta\}.$$

Then for the monoidal 2-tape automaton

$$\mathcal{A}' = \langle \mathcal{M}_2 \times \mathcal{M}_1, Q, I, F, \Delta' \rangle$$

we have $L(\mathcal{A}') = L(\mathcal{A})^{-1}$*.*

4. *(Identity relation) Let* $\mathcal{A} = \langle \mathcal{M}, Q, I, F, \Delta \rangle$ *be a monoidal automaton and*

$$\Delta' := \{\langle q_1, \langle u, u \rangle, q_2 \rangle \mid \langle q_1, u, q_2 \rangle \in \Delta\}.$$

Then for the monoidal 2-tape automaton

$$\mathcal{A}' = \langle \mathcal{M} \times \mathcal{M}, Q, I, F, \Delta' \rangle$$

we have $L(\mathcal{A}') = Id_{L(\mathcal{A})}$*.*

Corollary 4.2.2 *The class of monoidal multi-tape automaton relations is closed under Cartesian products, projections, and inverse relations.*

4.3 Classical multi-tape automata and letter automata

Definition 4.3.1 A *classical* n-tape automaton is an n-tape automaton over a monoid that is a Cartesian product of free monoids.

Definition 4.3.2 An *n-tape letter automaton* is a classical n-tape automaton $\mathcal{A} = \langle \Sigma_1 \times \Sigma_2 \times \ldots \times \Sigma_n, Q, I, F, \Delta \rangle$ such that $\Delta \subseteq Q \times (\Sigma_1^{\varepsilon} \times \ldots \times \Sigma_n^{\varepsilon}) \times Q$.

Proposition 4.3.3 *Let* $\mathcal{A}$ *be a classical* n*-tape automaton. Then* $\mathcal{A}$ *can be converted to an equivalent* n*-tape letter automaton.*

Proposition 4.3.4 *(Relational composition) Let*

$$\begin{aligned}\mathcal{A}_1 &= \langle \Sigma_1 \times \Sigma, Q_1, I_1, F_1, \Delta_1 \rangle \\ \mathcal{A}_2 &= \langle \Sigma \times \Sigma_2, Q_2, I_2, F_2, \Delta_2 \rangle\end{aligned}$$

be 2-tape letter automata. Let Δ be the set of all tuples

$$\langle \langle q_1, q_2 \rangle, \langle u, w \rangle, \langle q_1', q_2' \rangle \rangle$$

where there exists $v \in \Sigma \cup \{\varepsilon\}$ such that $\langle q_1, \langle u, v \rangle, q_1' \rangle \in E(\Delta_1)$ and $\langle q_2, \langle v, w \rangle, q_2' \rangle \in E(\Delta_2)$. Then for the 2-tape automaton

$$\mathcal{A} := \langle \Sigma_1 \times \Sigma_2, Q_1 \times Q_2, I_1 \times I_2, F_1 \times F_2, \Delta \rangle$$

we have $L(\mathcal{A}) = L(\mathcal{A}_1) \circ L(\mathcal{A}_2)$ (here $\circ$ denotes relational composition).

Remark 4.3.7 In [Ganchev et al., 2008] the n-tape one-letter automata are introduced which can be considered as n-tape letter automata for which the labels have ε on all tapes but one. All properties in this section hold for n-tape one-letter automata as well.

4.4 Monoidal finite-state transducers

Definition 4.4.1 A *monoidal finite-state transducer* is a monoidal 2-tape automaton $\mathcal{T} = \langle \Sigma^* \times \mathcal{M}, Q, I, F, \Delta \rangle$ where the underlying product monoid has the form $\Sigma^* \times \mathcal{M}$ for some alphabet Σ. If $\Delta \subseteq Q \times (\Sigma^\varepsilon \times \mathcal{M}) \times Q$, then $\mathcal{T}$ is called a monoidal *letter transducer*.

Definition 4.4.4 A monoidal finite-state transducer $\mathcal{T} = \langle \Sigma^* \times \mathcal{M}, Q, I, F, \Delta \rangle$ is *functional* iff the language $L(\mathcal{T})$ represents a partial function $\Sigma^* \to \mathcal{M}$. In this situation, $\mathcal{T}$ is said to *represent* the function $L(\mathcal{T})$.

Definition 4.4.5 A monoidal finite-state transducer $\mathcal{T} = \langle \Sigma^* \times \mathcal{M}, Q, I, F, \Delta \rangle$ is *infinitely ambiguous* if the relation $L(\mathcal{T})$ is infinitely ambiguous (cf. Definition 1.1.6).

Definition 4.4.6 A monoidal finite-state transducer $\mathcal{T} = \langle \Sigma^* \times \mathcal{M}, Q, I, F, \Delta \rangle$ is said to be *real-time* if $\Delta \subseteq Q \times (\Sigma \times \mathcal{M}) \times Q$.

Proposition 4.4.8 *Let $\mathcal{T} = \langle \Sigma^* \times \mathcal{M}, Q, I, F, \Delta \rangle$ be a monoidal letter transducer. Assume that the set of all path labels of the form $\langle \varepsilon, m \rangle$ in $\mathcal{T}$ is finite. Then there exists a real-time transducer $\mathcal{T}'$ equivalent to $\mathcal{T}$ up to ε.*

4.5 Classical finite-state transducers

Definition 4.5.1 A *classical* finite-state transducer is a transducer

$$\mathcal{T} = \langle \Sigma_1^* \times \Sigma_2^*, Q, I, F, \Delta \rangle$$

where the second tape runs over a free monoid Σ_2^*.

Proposition 4.5.7 *A trimmed classical finite-state transducer $\mathcal{T}$ is infinitely ambiguous iff there exists a loop in $\mathcal{T}$ with a label $\langle \varepsilon, u \rangle$, where $u \neq \varepsilon$.*

4.6 Deciding functionality of classical finite-state transducers

Definition 4.6.1 Let Σ be a finite alphabet. The *advance function* $\omega : (\Sigma^* \times \Sigma^*) \times (\Sigma^* \times \Sigma^*) \to \Sigma^* \times \Sigma^*$ is defined as:

$$\omega(\langle x, y \rangle, \langle \alpha, \beta \rangle) = \left\langle c^{-1}x\alpha, c^{-1}y\beta \right\rangle, \text{ where } c = x\alpha \wedge y\beta.$$

The *iterated advance function* $\omega^* : (\Sigma^* \times \Sigma^*) \times (\Sigma^* \times \Sigma^*)^* \to (\Sigma^* \times \Sigma^*)$ is defined inductively (note that ε and U denote sequences of pairs of strings):

- $\omega^*(\langle x, y \rangle, \varepsilon) = \langle x, y \rangle$,
- $\omega^*(\langle x, y \rangle, U \langle \alpha, \beta \rangle) = \omega(\omega^*(\langle x, y \rangle, U), \langle \alpha, \beta \rangle)$.

We say that $\omega(\langle x, y \rangle, \langle \alpha, \beta \rangle)$ (resp. $\omega^*(\langle x, y \rangle, U)$) is the *advance* of $\langle x, y \rangle$ with $\langle \alpha, \beta \rangle$ (resp. U).
The advance $\omega(\langle x, y \rangle, \langle \alpha, \beta \rangle) = \langle u, v \rangle$ of the pair of words $\langle x, y \rangle \in \Sigma^* \times \Sigma^*$ with $\langle \alpha, \beta \rangle \in \Sigma^* \times \Sigma^*$ is said to be *balancible* if $u = \varepsilon$ or $v = \varepsilon$.

Definition 4.6.4 Let $\mathcal{T} = \langle \Sigma_I^* \times \Sigma^*, Q, I, F, \Delta \rangle$ be a classical real-time finite-state transducer ($\Delta \subseteq Q \times (\Sigma_I \times \Sigma^*) \times Q$). The *squared output transducer* of $\mathcal{T}$ is the classical 2-tape automaton

$$\mathcal{S}_\mathcal{T} = \langle \Sigma^* \times \Sigma^*, Q \times Q, I \times I, F \times F, \Delta' \rangle$$

where Δ' is the set of all transitions of the form

$$\langle \langle q_1', q_2' \rangle, \langle \alpha_1, \alpha_2 \rangle, \langle q_1'', q_2'' \rangle \rangle$$

where there exists $a \in \Sigma$ such that $\langle q_1', \langle a, \alpha_1 \rangle, q_1'' \rangle \in \Delta$, and $\langle q_2', \langle a, \alpha_2 \rangle, q_2'' \rangle \in \Delta$.

Proposition 4.6.5 *Let $\mathcal{T} = \langle \Sigma_I^* \times \Sigma^*, Q, I, F, \Delta \rangle$ be a classical real-time finite-state transducer $\mathcal{T} = \langle \Sigma_I^* \times \Sigma^*, Q, I, F, \Delta \rangle$. Then $\mathcal{T}$ is functional iff for each successful path with label $\langle \alpha, \beta \rangle$ in the squared output transducer $\mathcal{S}_\mathcal{T}$ we have $\alpha = \beta$.*

Definition 4.6.6 Let $\mathcal{T} = \langle \Sigma^* \times \Sigma^*, Q, I, F, \Delta \rangle$ be a classical finite-state transducer. The pair $\langle u, v \rangle \in \Sigma^* \times \Sigma^*$ is called an *admissible advance* of the state $q \in Q$ if there exists a path

$$\pi : q_0 \to^{\langle \alpha_1, \beta_1 \rangle} q_1 \to^{\langle \alpha_2, \beta_2 \rangle} q_2 \ldots \to^{\langle \alpha_n, \beta_n \rangle} q_n = q$$

starting from an initial state $q_0 \in I$ such that

$$\langle u, v \rangle = \omega^*(\langle \varepsilon, \varepsilon \rangle, \langle \alpha_1, \beta_1 \rangle \ldots \langle \alpha_n, \beta_n \rangle).$$

By $Adm(q)$ we denote the *set of all admissible advances* of the state $q \in Q$.

Corollary 4.6.7 *Let $\mathcal{T} = \langle \Sigma_I^* \times \Sigma^*, Q, I, F, \Delta \rangle$ be a classical real-time finite-state transducer. Let $\mathcal{S}_\mathcal{T}$ be the squared output transducer of $\mathcal{T}$, with set of final states $F' = F \times F$. Then $\mathcal{T}$ is functional iff for each $p \in F'$ we have $Adm(p) \subseteq \{\langle \varepsilon, \varepsilon \rangle\}$.*

Proposition 4.6.8 *Let $\mathcal{T} = \langle \Sigma_I^* \times \Sigma^*, Q, I, F, \Delta \rangle$ be a classical real-time finite-state transducer, let $\mathcal{S}_\mathcal{T}$ be the squared output transducer of $\mathcal{T}$, let p be a state on a successful path of $\mathcal{S}_\mathcal{T}$. Assume that*

1. *there exists $\langle u, v \rangle \in Adm(p)$ such that $\langle u, v \rangle$ is not balancible (i.e., $u \neq \varepsilon$ and $v \neq \varepsilon$), or*
2. *$|Adm(p)| > 1$.*

Then $\mathcal{T}$ is not functional.

Proposition 4.6.10 *Let $\mathcal{T}$ be a classical real-time finite-state transducer and let $\mathcal{S}_\mathcal{T} = \langle \Sigma^* \times \Sigma^*, Q \times Q, I \times I, F \times F, \Delta' \rangle$ be the squared output transducer of $\mathcal{T}$. Then the pair $\langle u, v \rangle \in \Sigma^* \times \Sigma^*$ is an admissible advance of $q \in Q \times Q$ iff*

- *$q \in I \times I$ and $\langle u, v \rangle = \langle \varepsilon, \varepsilon \rangle$, or*
- *there exist a state $q' \in Q \times Q$, an admissible advance $\langle u', v' \rangle$ of q' and a transition $\langle q', \langle \alpha, \beta \rangle, q \rangle \in \Delta'$ such that $\langle u, v \rangle = \omega(\langle u, v \rangle, \langle \alpha, \beta \rangle)$.*

Corollary 4.6.11 *Let $\mathcal{T}$ be a classical real-time finite-state transducer and let $\mathcal{S}_{\mathcal{T}} = \langle \Sigma^* \times \Sigma^*, Q \times Q, I \times I, F \times F, \Delta' \rangle$ be the squared output transducer of $\mathcal{T}$. Let the functions $Adm^{(k)} : Q \times Q \to 2^{\Sigma^* \times \Sigma^*}$ be defined inductively:*

1. $Adm^{(0)}(q) := \begin{cases} \{\langle \varepsilon, \varepsilon \rangle\} & \text{if } q \in I \times I \\ \emptyset & \text{otherwise.} \end{cases}$

2. $Adm^{(k+1)}(q) := Adm^{(k)}(q) \cup$
   $\{\{\omega(\langle u', v' \rangle, \langle \alpha, \beta \rangle)\} \mid \langle q', \langle \alpha, \beta \rangle, q \rangle \in \Delta', \langle u', v' \rangle \in Adm^{(k)}(q')\}.$

Then $Adm = \bigcup_{k=0}^{\infty} Adm^{(k)}$.

Corollary 4.6.12 *Let $\mathcal{T}$ be a classical finite-state transducer. Then T is functional iff*

1. $|(\{\varepsilon\} \times \Sigma^*) \cap L(\mathcal{T})| \leq 1$;
2. *$\mathcal{T}$ is not infinitely ambiguous;*
3. *$\mathcal{T}'$ is functional, where $\mathcal{T}'$ is the classical real-time finite-state transducer equivalent to $\mathcal{T}$ up to ε.*

The propositions above provide us with a procedure for deciding the functionality of a given classical finite-state transducer, implemented in Program 8.2.12.

5 Deterministic transducers

In this chapter we explore deterministic finite-state transducers. Obviously, it only makes sense to ask for determinism if we restrict attention to transducers with a functional input-output behaviour. In this chapter we focus on transducers that are deterministic on the input tape (called sequential or subsequential transducers). The subsequential finite-state transducers are widely used for text processing [Mohri, 1996, Roche and Schabes, 1997b] and speech processing [Mohri et al., 2008].

5.1 Deterministic transducers and subsequential transducers

Definition 5.1.1 A monoidal finite-state transducer $\mathcal{T} = \langle \Sigma^* \times \mathcal{M}, Q, I, F, \Delta \rangle$ is *deterministic* if the following conditions hold:

1. $|I| = 1$, i.e., there is exactly one initial state;

2. $\delta := \{\langle q_1, a, q_2\rangle \mid \exists m \in M : \langle q_1, \langle a, m\rangle, q_2\rangle \in \Delta\}$ is a (partial) function $Q \times \Sigma \to Q$;

3. $\lambda := \{\langle q_1, a, m\rangle \mid \exists q_2 \in Q : \langle q_1, \langle a, m\rangle, q_2\rangle \in \Delta\}$ is a (partial) function $Q \times \Sigma \to M$;

The functions δ and λ are respectively called the *transition function* and the *transition output function*, and Δ is called the *transition relation*. Note that δ and λ have the same domain since both are derived from Δ. Deterministic monoidal finite-state transducers are also denoted in the form

$$\mathcal{T} = \langle \Sigma, \mathcal{M}, Q, q_0, F, \delta, \lambda\rangle$$

where $I = \{q_0\}$ and $\Delta = \{\langle q, \langle a, \lambda(q,a)\rangle, \delta(q,a)\rangle \mid \langle q, a\rangle \in dom(\delta)\}$.

Classical deterministic finite-state transducers are defined accordingly, demanding that the monoid $\mathcal{M}$ is free.

Definition 5.1.4 A *monoidal subsequential transducer* is a tuple

$$\mathcal{T} = \langle \Sigma, \mathcal{M}, Q, q_0, F, \delta, \lambda, \Psi\rangle$$

where $\langle \Sigma, \mathcal{M}, Q, q_0, F, \delta, \lambda\rangle$ is a deterministic monoidal finite-state transducer and $\Psi : F \to \mathcal{M}$ is the *state output function* with domain F. The *underlying automaton* of $\mathcal{T}$ is the deterministic finite-state automaton $\mathcal{A}_\mathcal{T} = \langle \Sigma, Q, q_0, F, \delta\rangle$ and the *input language* of $\mathcal{T}$ is the set

$$L(\mathcal{T})_{\times 2} = L(\mathcal{A}_\mathcal{T}) = \{t \in \Sigma^* \mid \delta^*(q_0, t) \in F\}.$$

A *classical subsequential transducer* is a monoidal subsequential transducer where the monoid $\mathcal{M}$ is the free monoid Σ'^* over a finite *output alphabet* Σ'. Classical subsequential transducer are denoted also with $\langle \Sigma, \Sigma', Q, q_0, F, \delta, \lambda, \Psi\rangle$.

Definition 5.1.5 Let $\mathcal{T} = \langle \Sigma, \mathcal{M}, Q, q_0, F, \delta, \lambda, \Psi\rangle$ be a monoidal subsequential transducer. The *output function* $O_\mathcal{T} : L(\mathcal{T})_{\times 2} \to \mathcal{M}$ of $\mathcal{T}$ is defined as follows ("$\cdot$" represents the monoid operation):

$$\forall t \in L(\mathcal{T})_{\times 2} : \qquad O_\mathcal{T}(t) := \lambda^*(q_0, t) \cdot \Psi(\delta^*(q_0, t)).$$

Definition 5.1.7 Let $\mathcal{T} = \langle \Sigma, \mathcal{M}, Q, q_0, F, \delta, \lambda, \Psi\rangle$ be a monoidal subsequential transducer. The *output function for* $q \in Q$ is

$$O^q_\mathcal{T} : \Sigma^* \to \mathcal{M};\ \alpha \mapsto \lambda^*(q, \alpha) \cdot \Psi(\delta^*(q, \alpha)).$$

Proposition 5.1.10 *Let* $\mathcal{T}_1 = \langle \Sigma, \Sigma', Q_1, q_{01}, F_1, \delta_1, \lambda_1, \Psi_1 \rangle$ *be a classical subsequential transducer and* $\mathcal{T}_2 = \langle \Sigma', \mathcal{M}, Q_2, q_{02}, F_2, \delta_2, \lambda_2, \Psi_2 \rangle$ *be a monoidal subsequential transducer. Then for the transducer*

$$\mathcal{T} = \langle \Sigma, \mathcal{M}, Q_1 \times Q_2, \langle q_{01}, q_{02} \rangle, F, \delta, \lambda, \Psi \rangle$$

where

- $\delta = \{\langle \langle q_1, q_2 \rangle, \sigma, \langle \delta_1(q_1, \sigma), \delta_2^*(q_2, \lambda_1(q_1, \sigma)) \rangle \rangle \mid q_1 \in Q_1, q_2 \in Q_2, \sigma \in \Sigma\}$,
- $\lambda = \{\langle \langle q_1, q_2 \rangle, \sigma, \lambda_2^*(q_2, \lambda_1(q_1, \sigma)) \rangle \mid q_1 \in Q_1, q_2 \in Q_2, \sigma \in \Sigma\}$,
- $F = \{\langle q_1, q_2 \rangle \mid q_1 \in F_1, \delta_2^*(q_2, \Psi_1(q_1)) \in F_2\}$,
- $\Psi = \{\langle \langle q_1, q_2 \rangle, \lambda_2^*(q_2, \Psi_1(q_1)) \odot \Psi_2(\delta_2^*(q_2, \Psi_1(q_1))) \rangle \mid \langle q_1, q_2 \rangle \in F\}$;

we have $O_{\mathcal{T}} = O_{\mathcal{T}_1} \circ O_{\mathcal{T}_2}$.

Definition 5.1.12 The *sequential distance* of $u \in \Sigma^*$ and $v \in \Sigma^*$ is defined as $d_S(u, v) = |u| + |v| - 2|u \wedge v|$.

Definition 5.1.13 A regular string function $f : \Sigma^* \to \Sigma'^*$ has the *bounded variation property* iff for all $k \geq 0$ there exists $K \geq 0$ such that for all $u, v \in dom(f)$ always $d_S(u, v) \leq k$ implies $d_S(f(u), f(v)) \leq K$.

The definition roughly says that two input strings that are identical up to small suffixes are translated into output strings that are similar up to suffixes.

Definition 5.1.14 A functional classical transducer $\mathcal{T}$ has the *bounded variation property* iff the regular string function represented by $\mathcal{T}$ has the bounded variation property.

5.2 A determinization procedure for functional transducers with the bounded variation property

Construction
We assume that a trimmed real-time functional classical transducer

$$\mathcal{T} = \langle \Sigma^* \times \Sigma'^*, Q, I, F, \Delta \rangle$$

over $\Sigma \times \Sigma'^*$ is given. I.e. $\Delta \subseteq Q \times (\Sigma \times \Sigma'^*) \times Q$.
The subsequential transducer

$$\mathcal{T}' = \langle \Sigma, \Sigma', Q', q_0', F', \delta', \lambda', \Psi' \rangle$$

equivalent to $\mathcal{T}$ is built by induction.

The base of the induction is:

$$\mathcal{T}'^{(0)} = \langle \Sigma, \Sigma', \{q'_0\}, q'_0, \emptyset, \emptyset, \emptyset, \emptyset \rangle \text{ , where } q'_0 = I \times \{\varepsilon\}.$$

Let us assume that we have constructed

$$\mathcal{T}'^{(n)} = \left\langle \Sigma, \Sigma', Q'^{(n)}, q'_0, F'^{(n)}, \delta'^{(n)}, \lambda'^{(n)}, \Psi'^{(n)} \right\rangle .$$

We define $\mathcal{T}'^{(n+1)} = \langle \Sigma, \Sigma', Q'^{(n+1)}, q'_0, F'^{(n+1)}, \delta'^{(n+1)}, \lambda'^{(n+1)}, \Psi'^{(n+1)} \rangle$ with the following components:

- The new transition output function $\lambda'^{(n+1)}$ extends $\lambda'^{(n)}$ with all triples of the form $\langle S, \sigma, w \rangle$ for which $\{\langle q, \langle \sigma, v \rangle, q' \rangle \in \Delta \mid \langle q, u \rangle \in S\} \neq \emptyset$, where $S \in Q'^{(n)}$, $\sigma \in \Sigma$ and

$$w = \bigwedge_{\langle q,u\rangle \in S} \bigwedge_{\langle q,\langle \sigma,v\rangle,q'\rangle \in \Delta} u \cdot v.$$

- The new transition function $\delta'^{(n+1)}$ extends $\delta'^{(n)}$ with all triples of the form $\langle S, \sigma, S' \rangle$ where $\langle S, \sigma, w \rangle$ is a triple in $\lambda'^{(n+1)}$ and

$$S' = \bigcup_{\langle q,u\rangle \in S} \bigcup_{\langle q,\langle \sigma,v\rangle,q'\rangle \in \Delta} \{\langle q', w^{-1}(u \cdot v)\rangle\}$$

- $Q'^{(n+1)} = Q'^{(n)} \cup codom(\delta'^{(n+1)})$,
- $F'^{(n+1)} = \{S \in Q'^{(n+1)} \mid \exists \langle q, \beta \rangle \in S \ : \ q \in F\}$,
- $\Psi'^{(n+1)} = \{\langle S, \beta \rangle \mid S \in F'^{(n+1)}, \exists \langle q, \beta \rangle \in S \ : \ q \in F\}$.

Lemma 5.2.2 *Let $\mathcal{T}$ be as above. Let $\mathcal{T}'^{(n)} = \langle \Sigma, \Sigma', Q', q'_0, F', \delta', \lambda', \Psi' \rangle$ be constructed by the induction given above in n steps. Then for each word $w \in \Sigma^*$ such that $\lambda'^*(q'_0, w)$ and $\delta'^*(q'_0, w)$ are defined we have the following properties:*

$$\begin{aligned} \lambda'^*(q'_0, w) &= \bigwedge_{q_0 \in I, \langle q_0, \langle w, u\rangle, q\rangle \in \Delta^*} u \\ \delta'^*(q'_0, w) &= \{\langle q, \gamma \rangle \mid \exists q_0 \in I \ \exists \langle q_0, \langle w, u \rangle, q \rangle \in \Delta^* \ : \ \gamma = \lambda'^*(q'_0, w)^{-1} u\}. \end{aligned}$$

Lemma 5.2.3 *Let $\mathcal{T}'^{(n)} = \langle \Sigma, \Sigma', Q', q'_0, F', \delta', \lambda', \Psi' \rangle$ be constructed by the induction given above in n steps from the transducer $\mathcal{T} = \langle \Sigma^* \times \Sigma'^*, Q, \{q_0\}, F, \Delta \rangle$. Then for each state $S \in Q'$ and $q \in Q$ we have $|\{v \in \Sigma'^* \mid \exists \langle q, v \rangle \in S\}| \leq 1$.*

Lemma 5.2.4 *Let $\mathcal{T}'^{(n)} = \langle \Sigma, \Sigma', Q', q'_0, F', \delta', \lambda', \Psi' \rangle$ be constructed by the induction given above in n steps from the transducer $\mathcal{T} = \langle \Sigma^* \times \Sigma'^*, Q, \{q_0\}, F, \Delta \rangle$. Then for each state $S \in Q'$ and $\langle q_1, v_1 \rangle, \langle q_2, v_2 \rangle \in S$ we have*

$$q_1 \in F \ \& \ q_2 \in F \rightarrow v_1 = v_2.$$

Proposition 5.2.5 *Let $\mathcal{T} = \langle \Sigma^* \times \Sigma'^*, Q, I, F, \Delta \rangle$ be a functional real-time classical transducer such that the inductive construction of $\mathcal{T}'$ presented above terminates in the sense that there exists a number $k \in \mathbb{N}$ such that $\mathcal{T}'^{(k)} = \mathcal{T}'^{(k+1)} = \mathcal{T}'^{(k+2)} = \ldots = \mathcal{T}'$, let $\mathcal{T}' = \langle \Sigma, \Sigma', Q', q'_0, F', \delta', \lambda', \Psi' \rangle$. Then $O_{\mathcal{T}'} = L(\mathcal{T})$.*

Theorem 5.2.6 *Let $\mathcal{T} = \langle \Sigma^* \times \Sigma'^*, Q, I, F, \Delta \rangle$ be a trimmed real-time functional classical transducer with the bounded variation property. Then the inductive construction of $\mathcal{T}'$ presented above terminates in the sense that there exists a number $k \in \mathbb{N}$ such that $\mathcal{T}'^{(k)} = \mathcal{T}'^{(k+1)} = \mathcal{T}'^{(k+2)} = \ldots = \mathcal{T}'$.*

Corollary 5.2.7 *A regular string function $f : \Sigma^* \rightarrow \Sigma'^*$ can be represented by a classical subsequential transducer iff f has the bounded variation property.*

5.3 Deciding the bounded variation property

Lemma 5.3.3 *Let $\mathcal{T}$ be a classical real-time finite-state transducer, let q denote a state of the squared output transducer $\mathcal{S}_{\mathcal{T}}$ of $\mathcal{T}$.*

1. *If $\langle u, v \rangle \in \mathrm{Adm}(q)$ and the corresponding path in $\mathcal{S}_{\mathcal{T}}$ is $\pi = q_0 \rightarrow \ldots \overset{\langle \alpha, \beta \rangle}{\rightarrow} q$, then $d_S(\alpha, \beta) = |u| + |v|$.*

2. *If $\mathrm{Adm}(q)$ is finite and there exists $\langle u, v \rangle \in \mathrm{Adm}(q)$ such that $\langle u, v \rangle$ is not balancible, then each loop $\langle q, \langle \alpha, \beta \rangle, q \rangle \in \Delta'^*$ has label $\langle \alpha, \beta \rangle = \langle \varepsilon, \varepsilon \rangle$.*

Theorem 5.3.4 *Let $\mathcal{T} = \langle \Sigma_I^* \times \Sigma^*, Q, I, F, \Delta \rangle$ be a trimmed functional classical real-time finite-state transducer and let $\mathcal{S}_{\mathcal{T}}$ be the squared output transducer of $\mathcal{T}$. Then $\mathcal{T}$ has the bounded variation property iff for each state q of $\mathcal{S}_{\mathcal{T}}$ the set of admissible advances $\mathrm{Adm}(q)$ is finite.*

Lemma 5.3.5 *Let $\mathcal{T} = \langle \Sigma_I^* \times \Sigma^*, Q, I, F, \Delta \rangle$ be a trimmed functional classical real-time finite-state transducer, let $C := \max\limits_{\langle q', \langle \sigma, \alpha \rangle, q'' \rangle \in \Delta} |\alpha|$. Let $\langle u, v \rangle$ be an admissible advance of the state $\langle p, q \rangle \in Q \times Q$ of the squared output transducer $\mathcal{S}_{\mathcal{T}}$. If $\mathcal{T}$ has the bounded variation property, then $|u| < C|Q|^2$ and $|v| < C|Q|^2$.*

The above propositions provide us with an effective procedure for deciding the bounded variation property of a classical real-time transducer, implemented in Program 8.3.2.

Corollary 5.3.7 *Let $\mathcal{T} = \langle \Sigma_I^* \times \Sigma^*, Q, I, F, \Delta \rangle$ be a trimmed functional classical real-time finite-state transducer, let $\mathcal{S}_\mathcal{T} = \langle \Sigma^* \times \Sigma^*, Q \times Q, I \times I, F \times F, \Delta' \rangle$ be the squared output transducer of $\mathcal{T}$. Let $\mathcal{T}'^{(n)} = \langle \Sigma_I, \Sigma^*, Q', q_0', F', \delta', \lambda', \Psi' \rangle$ be constructed in n steps by the inductive determinization procedure described in Section 5.2. Let $S \in Q'$ be a state in $\mathcal{T}'^{(n)}$. Then for each pair $\langle p, u \rangle \in S$ there exists a pair $\langle q, v \rangle \in S$ such that $\langle u, v \rangle$ is an admissible advance of the state $\langle p, q \rangle$ (i.e $\langle u, v \rangle \in Adm(\langle p, q \rangle)$).*

Theorem 5.3.8 *Let $\mathcal{T} = \langle \Sigma^* \times \Sigma^*, Q, I, F, \Delta \rangle$ be a trimmed real-time functional classical transducer, let $C := \max_{\langle q', \langle \sigma, \alpha \rangle, q'' \rangle \in \Delta} |\alpha|$. Then the inductive construction presented in Section 5.2 terminates iff at each step n of the construction of the corresponding subsequential transducer*

$$\mathcal{T}'^{(n)} = \langle \Sigma_I, \Sigma^*, Q', q_0', F', \delta', \lambda', \Psi' \rangle$$

for each state S in Q' and each pair $\langle p, u \rangle \in S$ we have $|u| < C|Q|^2$.

he above theorem provides us with an effective way for testing the termination of the inductive determinization construction presented in Section 5.2.

5.4 Minimal subsequential finite-state transducers - Myhill-Nerode relation for subsequential transducers

We now show that also in the case of subsequential finite-state transducers a new kind of Myhill-Nerode relation yields a minimal subsequential transducer.

Definition 5.4.1 Let $f : \Sigma^* \to \Sigma'^*$ be a (partial) function. Then

$$\begin{aligned} R_f = \{\langle u, v \rangle \in \Sigma^* \times \Sigma^* \mid \; & \exists u' \in \Sigma'^* \; \exists v' \in \Sigma'^* \quad \forall w \in \Sigma^* : \\ & (u \cdot w \in dom(f) \leftrightarrow \quad v \cdot w \in dom(f)) \; \& \\ & (u \cdot w \in dom(f) \rightarrow \quad u' \in Pref(f(u \cdot w)) \; \& \; v' \in Pref(f(v \cdot w)) \; \& \\ & \qquad\qquad u'^{-1} f(u \cdot w) = v'^{-1} f(v \cdot w))\} \end{aligned}$$

is called the *Myhill-Nerode relation* for f.

Proposition 5.4.2 *Let $f : \Sigma^* \to \Sigma'^*$ be a function. Then the Myhill-Nerode relation for f is a right invariant equivalence relation.*

Definition 5.4.3 Let $f : \Sigma^* \to \Sigma'^*$ be a function. The *longest common output* of f is the function $lco_f : \Sigma^* \to \Sigma'^*$ defined as follows:

$$lco_f(u) = \begin{cases} \bigwedge_{w \in \Sigma^* \ \&\ u \cdot w \in dom(f)} f(u \cdot w) & \text{if } u \in Pref(dom(f)) \\ \varepsilon & \text{otherwise.} \end{cases}$$

Definition 5.4.4 A *subsequential finite-state transducer with initial output* over the monoid $\mathcal{M} = \langle M, \circ, e \rangle$ is a tuple $\mathcal{T} = \langle \Sigma, \mathcal{M}, Q, q_0, F, \delta, \lambda, \iota, \Psi \rangle$ such that $\langle \Sigma, \mathcal{M}, Q, q_0, F, \delta, \lambda, \Psi \rangle$ is a subsequential finite-state transducer and $\iota \in M$. The output function of the subsequential finite-state transducer with initial output $\mathcal{T}$ is defined as $O_{\mathcal{T}}(\alpha) = \iota \circ \lambda^*(q_0, \alpha) \circ \Psi(\delta^*(q_0, \alpha))$.

Proposition 5.4.6 *Let $\mathcal{T} = \langle \Sigma, \Sigma', Q, q_0, F, \delta, \lambda, \iota, \Psi \rangle$ be a classical subsequential finite-state transducer with initial output representing $f : \Sigma^* \to \Sigma'^*$ and total transition function δ. Let*

$$R_{\mathcal{T}} := \{\langle u, v \rangle \in \Sigma^* \times \Sigma^* \mid \delta^*(q_0, u) = \delta^*(q_0, v)\}.$$

Then $R_{\mathcal{T}}$ is a right invariant equivalence relation and $R_{\mathcal{T}}$ is a refinement of the Myhill-Nerode relation R_f.

Corollary 5.4.7 *Let $\mathcal{T} = \langle \Sigma, \Sigma', Q, q_0, F, \delta, \lambda, \iota, \Psi \rangle$ be a subsequential classical finite-state transducer with initial output and total transition function δ representing $f : \Sigma^* \to \Sigma'^*$. Then $|\Sigma^*/R_f| \leq |\Sigma^*/R_{\mathcal{T}}| = |Q|$.*

Proposition 5.4.8 *Let $f : \Sigma^* \to \Sigma'^*$ be a function such that the index of the Myhill-Nerode relation R_f is finite. Let*

$$\mathcal{T}_f = \left\langle \Sigma, \Sigma', \Sigma^*/R_f, [\varepsilon]_{R_f}, \{[s]_{R_f} \mid s \in dom(f)\}, \delta, \lambda, \iota, \Psi \right\rangle,$$

where:

- $\delta = \{\left\langle [u]_{R_f}, a, [u \cdot a]_{R_f} \right\rangle \mid u \in \Sigma^*, a \in \Sigma\}$,
- $\lambda = \{\left\langle [u]_{R_f}, a, lco_f(u)^{-1} lco_f(u \cdot a) \right\rangle \mid u \in \Sigma^*, a \in \Sigma, u \cdot a \in Pref(dom(f))\} \cup \{\left\langle [u]_{R_f}, a, \varepsilon \right\rangle \mid u \in \Sigma^*, a \in \Sigma, u \cdot a \notin Pref(dom(f))\}$,
- $\iota = lco_f(\varepsilon)$,
- $\Psi = \{\left\langle [u]_{R_f}, lco_f(u)^{-1} f(u) \right\rangle \mid u \in dom(f)\}$.

Then $\mathcal{T}_f$ is a classical subsequential finite-state transducer with initial output and we have $O_{\mathcal{T}_f} = f$ and the transition function δ is total.

Theorem 5.4.10 *Let $f : \Sigma^* \to \Sigma'^*$ be a function. Then f is the output function of a subsequential finite-state transducer iff the index of R_f is finite.*

Theorem 5.4.11 *Let $f : \Sigma^* \to \Sigma'^*$ be a function such that the index of R_f is finite. Then the Myhill-Nerode transducer $\mathcal{T}_f$ for f is minimal with respect to number of states among all subsequential classical finite-state transducers with initial output and total transition function representing f.*

5.5 Minimization of subsequential transducers

In this section we show how to minimize a given subsequential finite-state transducer, essentially following the approach in [Mohri, 2000]. We present the procedure for the case of classical subsequential finite-state transducers. But essentially the same technique can be used for other target monoids that satisfy some additional conditions [Gerdjikov and Mihov, 2017a].

Definition 5.5.1 Let $\mathcal{T} = \langle \Sigma, \Sigma', Q, q_0, F, \delta, \lambda, \iota, \Psi \rangle$ be a trimmed classical subsequential finite-state transducer with initial output. The *maximal state output* $mso_{\mathcal{T}}(q)$ for a state $q \in Q$ is defined as

$$mso_{\mathcal{T}}(q) := \bigwedge_{w \in \Sigma^* \ \& \ \delta^*(q,w) \in F} O^q_{\mathcal{T}}(w).$$

Definition 5.5.2 Let $\mathcal{T} = \langle \Sigma, \Sigma', Q, q_0, F, \delta, \lambda, \iota, \Psi \rangle$ be a trimmed classical subsequential finite-state transducer with initial output. Then the *canonical form* of $\mathcal{T}$ is the subsequential finite-state transducer with initial output

$$\mathcal{T}' := \langle \Sigma, \Sigma', Q, q_0, F, \delta, \lambda', \iota', \Psi' \rangle$$

where

- $\iota' := \iota \cdot mso_{\mathcal{T}}(q_0)$,
- $\Psi'(q) := mso_{\mathcal{T}}(q)^{-1}\Psi(q)$ for all $q \in F$,
- $\lambda'(q, \sigma) := mso_{\mathcal{T}}(q)^{-1}(\lambda(q, \sigma) \cdot mso_{\mathcal{T}}(\delta(q, \sigma)))$, for all $q \in Q, \sigma \in \Sigma$ such that $\delta(q, \sigma)$ is defined.

Proposition 5.5.8 *Let $\mathcal{T} = \langle \Sigma, \Sigma', Q, q_0, F, \delta, \lambda, \iota, \Psi \rangle$ be a trimmed classical subsequential transducer in canonical form representing the function f. Let $q_1, q_2 \in Q$ be two states. Then q_1 and q_2 are equivalent iff*

1. *$q_1 \in F$ iff $q_2 \in F$,*

2. *if* $q_1 \in F$, *then* $\Psi(q_1) = \Psi(q_2)$,

3. *for all* $\sigma \in \Sigma$: $\delta(q_1, \sigma)$ *is defined iff* $\delta(q_2, \sigma)$ *is defined. If both are defined, then* $\delta(q_1, \sigma)$ *and* $\delta(q_2, \sigma)$ *are equivalent and* $\lambda(q_1, \sigma) = \lambda(q_2, \sigma)$.

Lemma 5.5.9 *Let* $\mathcal{T} = \langle \Sigma, \Sigma', Q, q_0, F, \delta, \lambda, \iota, \Psi \rangle$ *be a trimmed classical subsequential finite-state transducer with initial output in canonical form representing the function* $f : \Sigma^* \to \Sigma'^*$. *Let* $u, v \in \Sigma^*$, *let* $\delta^*(q_0, u) = q_1$ *and* $\delta^*(q_0, v) = q_2$ *be defined. Then* $u \; R_f \; v$ *iff* q_1 *is equivalent to* q_2.

Theorem 5.5.10 *A classical subsequential finite-state transducer* $\mathcal{T}$ *with initial output in canonical form representing the string function* f *is minimal (in terms of the number of states) among all classical subsequential finite-state transducers with initial output representing* f *iff* $\mathcal{T}$ *is trimmed and there are no distinct equivalent states in* $\mathcal{T}$.

Conversion to canonical form – computing maximal state outputs

Definition 5.5.11 Let $\mathcal{T} = \langle \Sigma, \Sigma', Q, q_0, F, \delta, \lambda, \iota, \Psi \rangle$ be a classical subsequential finite-state transducer with initial output. Then the monoidal finite-state automaton

$$\mathcal{A}_\mathcal{T} := \langle \Sigma'^*, Q \cup \{f\}, \{q_0\}, \{f\}, \Delta \rangle$$

where $f \notin Q$ is a new state and

$$\Delta = \{\langle q, \Psi(q), f \rangle \mid q \in F\} \cup \{\langle q', \lambda(q', \sigma), q'' \rangle \mid \langle q', \sigma, q'' \rangle \in \delta\}$$

is called the *output automaton* of $\mathcal{T}$.

Clearly, the language of $\mathcal{A}_\mathcal{T}$ is equal to $\iota^{-1} codom(O_\mathcal{T})$ and for each state $q \in Q$ we have

$$L_{\mathcal{A}_\mathcal{T}}(q) = \bigcup_{w \in \Sigma^* \;\&\; \delta^*(q,w) \in F} O^q_\mathcal{T}(w).$$

Following Proposition 3.2.1 we compute a classical finite-state automaton $\mathcal{A}'_\mathcal{T}$ with an extended set of states Q' without ε-transitions such that for each $q \in Q$ we have $L_{\mathcal{A}_\mathcal{T}}(q) = L_{\mathcal{A}'_\mathcal{T}}(q)$. $\mathcal{A}'_\mathcal{T}$ is called the *expanded output automaton* of $\mathcal{T}$.

Proposition 5.5.13 *Let* $\mathcal{T} = \langle \Sigma, \Sigma', Q, q_0, F, \delta, \lambda, \iota, \Psi \rangle$ *be a classical subsequential finite-state transducer with initial output, let*

$$\mathcal{A}'_\mathcal{T} = \langle \Sigma', Q \cup Q'' \cup \{f\}, q_0, F'', \Delta'' \rangle$$

be the expanded output automaton of $\mathcal{T}$, let $q \in Q$. Then

$$L_{\mathcal{A}'_{\mathcal{T}}}(q) = \bigcup_{w \in \Sigma^*, \delta^*(q,w) \in F} O^q_{\mathcal{T}}(w)$$

and

$$mso_{\mathcal{T}}(q) = \bigwedge L_{\mathcal{A}'_{\mathcal{T}}}(q).$$

Proposition 5.5.14 *Let $\mathcal{A} = \langle \Sigma', Q, q_0, F, \delta \rangle$ be a trimmed deterministic finite-state automaton. Let*

$$\pi = q_0 \rightarrow^{a_1} q_1 \rightarrow^{a_2} q_2 \ldots q_{k-1} \rightarrow^{a_k} q_k$$

be a path starting at q_0. Then $w = \bigwedge L(\mathcal{A})$ for $w = a_1 a_2 \ldots a_k$ iff the following properties hold:

- *q_k is final or there are more than one outgoing transitions from q_k. i.e.*

$$q_k \in F \vee |\{\sigma \in \Sigma' \mid !\delta(q_k, \sigma)\}| > 1.$$

- *For each i in $0, \ldots, k-1$ the state q_i is not final and there is exactly one outgoing transition from q_i. I.e.*

$$\forall i \in \{0, \ldots, k-1\} : q_i \notin F \ \& \ |\{\sigma \in \Sigma' \mid !\delta(q_i, \sigma)\}| = 1.$$

Corollary 5.5.15 *Let $\mathcal{T} = \langle \Sigma, \Sigma', Q, q_0, F, \delta, \lambda, \iota, \Psi \rangle$ be a trimmed classical subsequential finite-state transducer with initial output and let*

$$\mathcal{A}'_{\mathcal{T}} = \langle \Sigma', Q \cup Q'' \cup \{f\}, q_0, F'', \Delta'' \rangle$$

be the expanded output automaton of $\mathcal{T}$. Let $\mathcal{D}^q = \langle \Sigma', Q_q, \{q\}, F_q, \delta_q \rangle$ be the deterministic finite-state automaton obtained from the determinization of ${\mathcal{A}'_{\mathcal{T}}}^q = \langle \Sigma', Q \cup Q'' \cup \{f\}, q, F'', \Delta'' \rangle$. Then

$$mso_{\mathcal{T}}(q) = \bigwedge L(\mathcal{D}^q) = w_q,$$

where $w_q \in \Sigma'^*$ is the label of the maximal unique path of $\mathcal{D}^q$.

Remark 5.5.16 In order to find the longest common prefix of the language of ${\mathcal{A}'_{\mathcal{T}}}^q$ (cf. Corollary 5.5.15) we can proceed by determinizing only the initial part of the automaton, until we reach the state q_k for which the conditions in Proposition 5.5.14 are fulfilled. Hence only a small part of the automaton has to be determinized. The complexity of finding $mso_{\mathcal{T}}(q)$ is $O(|w_q||Q \cup Q''|^2)$ (cf. Program 8.3.5).

Pseudo-minimization – computing the minimal subsequential transducer

Proposition 5.5.18 *Let $\mathcal{T} = \langle \Sigma, \Sigma', Q, q_0, F, \delta, \lambda, \iota, \Psi \rangle$ be a canonical trimmed subsequential finite-state transducer with initial output. Consider the new alphabet*

$$\Gamma := \{ \langle c, \lambda(q,c) \rangle \mid c \in \Sigma, q \in Q \ \& \ !\delta(q,c) \},$$

and the set of transitions $\delta_{\mathcal{A}} := \{ \langle q, \langle c, \lambda(q,c) \rangle, q' \rangle \mid \langle q, c, q' \rangle \in \delta \}$. Then

$$\mathcal{A}_{\mathcal{T}} := \langle \Gamma, codom(\Psi), Q, q_0, F, \delta_{\mathcal{A}}, \Psi \rangle$$

is a $codom(\Psi)$-coloured deterministic finite-state automaton. Let

$$\mathcal{A}'_{\mathcal{T}} = \langle \Gamma, codom(\Psi), Q', q'_0, F', \delta'_{\mathcal{A}}, \Psi' \rangle$$

denote the minimal $codom(\Psi)$-coloured deterministic finite-state automaton equivalent to $\mathcal{A}_{\mathcal{T}}$. Then the subsequential finite-state transducer

$$\mathcal{T}' := \langle \Sigma, \Sigma', Q', q'_0, F', \delta', \lambda', \iota, \Psi' \rangle,$$

where

$$\begin{aligned} \delta' &:= \{ \langle q, c, q' \rangle \mid \langle q, \langle c, \alpha \rangle, q' \rangle \in \delta'_{\mathcal{A}} \} \\ \lambda' &:= \{ \langle q, c, \alpha \rangle \mid \langle q, \langle c, \alpha \rangle, q' \rangle \in \delta'_{\mathcal{A}} \} \end{aligned}$$

is the minimal canonical subsequential finite-state transducer equivalent to $\mathcal{T}$.

Remark 5.5.20 For finite word functions there exist direct methods for constructing the minimal finite-state subsequential transducer which provide better efficiency [Mihov and Maurel, 2001].

5.6 Numerical subsequential transducers

In this section we show how the results from the previous sections can be transferred to the monoid of natural numbers.

Definition 5.6.1 Let $\mathcal{T} = \langle \Sigma, \Sigma', Q, q_0, F, \delta, \lambda, \Psi \rangle$ be a classical subsequential transducer. Let $\phi : \Sigma'^* \to \mathcal{M}$ be a homomorphism between the monoids Σ'^* and $\mathcal{M} = \langle M, \bullet, e \rangle$. Then the monoidal subsequential transducer $\mathcal{T}_\phi = \langle \Sigma, \mathcal{M}, Q, q_0, F, \delta, \lambda_\phi, \Psi_\phi \rangle$, where

- $\lambda_\phi := \lambda \circ \phi$ (i.e. $\lambda_\phi(q, \sigma) := \phi(\lambda(q, \sigma))$),

- $\Psi_\phi := \Psi \circ \phi$ (i.e. $\Psi_\phi(q) := \phi(\Psi(q))$),

is said to be *obtained by mapping $\mathcal{T}$ with ϕ.*

Proposition 5.6.2 *Let $\mathcal{T}_\phi$ be obtained from the subsequential transducer $\mathcal{T}$ by mapping with ϕ, where $\phi : \Sigma'^* \to \mathcal{M}$ is a homomorphism between the monoids Σ'^* and $\mathcal{M}$. Then $O_{\mathcal{T}_\phi} = O_\mathcal{T} \circ \phi$.*

Proposition 5.6.3 *Let $\Sigma' = \{1\}$ and $\mathcal{N} = \langle \mathbb{N}, +, 0 \rangle$. Then the function $\varphi : \Sigma'^* \to \mathbb{N}$ defined as*

$$\varphi(1^n) := n, \text{ for any } n \in \mathbb{N}$$

is a monoidal isomorphism between the monoids Σ'^* and $\mathcal{N}$.

Proposition 5.6.4 *Let $\mathcal{T}$ be a classical subsequential transducer and let $\Sigma' = \{1\}$. Then $\mathcal{T}$ can be obtained by mapping $\mathcal{T}_\varphi$ with φ^{-1} i.e. $\mathcal{T} = \mathcal{T}_{\varphi\varphi^{-1}}$.*

Corollary 5.6.5 *All results for transducer determinization (Section 5.2), deciding functionality (Section 4.6), deciding bounded variation (Section 5.3), and transducer minimization (Section 5.4) obtained for classical subsequential transducers are transferred directly to monoidal subsequential transducers over the monoid $\mathcal{N}$.*

6 Bimachines

In this chapter we look at a more powerful concept. We introduce bimachines, a deterministic finite-state device that exactly represents the class of all regular string functions.

6.1 Basic definitions

Definition 6.1.1 A *monoidal bimachine* is a tuple $\mathcal{B} = \langle \mathcal{M}, \mathcal{A}_L, \mathcal{A}_R, \psi \rangle$ where

- $\mathcal{M} = \langle M, \circ, e \rangle$ is a monoid,
- $\mathcal{A}_L = \langle \Sigma, L, s_L, L, \delta_L \rangle$ and $\mathcal{A}_R = \langle \Sigma, R, s_R, R, \delta_R \rangle$ are deterministic finite-state automata called the *left* and *right automaton* of the bimachine;
- $\psi : (L \times \Sigma \times R) \to M$ is a partial function called the *output function.*

Note that all states of $\mathcal{A}_L$ and $\mathcal{A}_R$ are final. If $\mathcal{M}$ is a free monoid, then $\mathcal{B}$ is called a *classical bimachine*. In this case we simply assume that the output alphabet coincides with the alphabets of the left and right automata, Σ, and write $\mathcal{B} = \langle \mathcal{A}_L, \mathcal{A}_R, \psi \rangle$.

Definition 6.1.2 Let $\mathcal{M} = \langle M, \circ, e \rangle$ be a monoid, let $\mathcal{B} = \langle \mathcal{M}, \mathcal{A}_L, \mathcal{A}_R, \psi \rangle$ denote a monoidal bimachine, let Σ denote the alphabet of $\mathcal{A}_L$ and $\mathcal{A}_R$. Consider an input sequence $t = \sigma_1 \sigma_2 \cdots \sigma_n \in \Sigma^*$ $(n \geq 0)$ with letters σ_i $(i = 1, \ldots, n)$. If all the values $\delta_L^*(\sigma_1 \sigma_2 \cdots \sigma_i)$ and $\delta_R^*(\sigma_n \sigma_{n-1} \cdots \sigma_i)$ are defined $(1 \leq i \leq n)$ we obtain a pair of paths

$$\begin{array}{lccccccc} \pi_L: & l_0 \rightarrow^{\sigma_1} l_1 & \rightarrow \ldots & l_{i-1} \rightarrow^{\sigma_i} l_i & \rightarrow \ldots & l_{n-1} \rightarrow^{\sigma_n} l_n \\ \pi_R: & r_0 \leftarrow^{\sigma_1} r_1 & \leftarrow \ldots & r_{i-1} \leftarrow^{\sigma_i} r_i & \leftarrow \ldots & r_{n-1} \leftarrow^{\sigma_n} r_n \end{array}$$

where π_L is a path of the left automaton $\mathcal{A}_L$ starting from $l_0 := s_L$ and π_R is a path of the right automaton $\mathcal{A}_L$ (arrows indicate reading order) starting from $r_n := s_R$. If all outputs $\psi(l_{i-1}, \sigma_i, r_i)$ are defined $(1 \leq i \leq n)$, then we call (π_L, π_R) a *pair of successful paths* of $\mathcal{B}$ with label $\sigma_1 \sigma_2 \ldots \sigma_n$ and output

$$O_{\mathcal{B}}(t) := \psi(l_0, \sigma_1, r_1) \circ \psi(l_1, \sigma_2, r_2) \circ \ldots \circ \psi(l_{i-1}, \sigma_i, r_i) \circ \ldots \circ \psi(l_{n-1}, \sigma_n, r_n).$$

In the special case where $t = \varepsilon$ we have $O_{\mathcal{B}}(t) = e$. The partial function $O_{\mathcal{B}}$ is called the *output function* of the bimachine, or the *function represented by the bimachine*. If $O_{\mathcal{B}}(t) = m$ we say that the bimachine $\mathcal{B}$ *translates* t *into* m.

Definition 6.1.4 Let $\mathcal{B} = \langle \mathcal{M}, \mathcal{A}_L, \mathcal{A}_R, \psi \rangle$ denote a monoidal bimachine. The *generalized output function* ψ^* of $\mathcal{B}$ is inductively defined as follows:

- $\psi^*(l, \varepsilon, r) = e$ for all $l \in L, r \in R$;
- $\psi^*(l, t\sigma, r) = \psi^*(l, t, \delta_R(r, \sigma)) \circ \psi(\delta_L^*(l, t), \sigma, r)$ for $l \in L, r \in R, t \in \Sigma^*, \sigma \in \Sigma$.

6.2 Equivalence of regular string functions and classical bimachines

We follow the approach presented in [Gerdjikov et al., 2017].

Proposition 6.2.1 *For each monoidal bimachine* $\mathcal{B} = \langle \mathcal{M}, \mathcal{A}_L, \mathcal{A}_R, \psi \rangle$ *there exists a monoidal finite-state transducer* $\mathcal{A} = \langle \Sigma^* \times \mathcal{M}, Q, I, F, \Delta \rangle$*, such that* $O_{\mathcal{B}} = L(\mathcal{A})$.

Proof. [Construction] We construct the real-time transducer

$$\mathcal{A} := \langle \Sigma^* \times \mathcal{M}, L \times R, \{s_L\} \times R, L \times \{s_R\}, \Delta \rangle$$

where Δ contains all transitions $\langle l, r \rangle \to_m^\sigma \langle l', r' \rangle$ such that $\delta_L(l, \sigma) = l', \delta_R(r', \sigma) = r, \psi(l, \sigma, r') = m$.

Proposition 6.2.5 *Let $\mathcal{T} = \langle \Sigma^* \times \mathcal{M}, Q, I, F, \Delta \rangle$ be a monoidal trimmed functional real-time transducer with output in the monoid $\mathcal{M} = \langle M, \circ, e \rangle$ such that $\langle \varepsilon, e \rangle \in L(\mathcal{T})$. Then there exists a monoidal bimachine $\mathcal{B} = \langle \mathcal{M}, \mathcal{A}_L, \mathcal{A}_R, \psi \rangle$ such that $L(\mathcal{T}) = O_{\mathcal{B}}$.*

Proof. [Construction] The *right deterministic automaton* of the bimachine $\mathcal{A}_R = \langle \Sigma, Q_R, s_R, Q_R, \delta_R \rangle$ is defined as the result when applying the determinization procedure to the reversed underlying automaton of $\mathcal{T}$ and setting all states to final. This means that $Q_R \subseteq 2^Q$, $s_R = F$ and

$$\delta_R(R, a) := \{q \in Q \mid \exists q' \in R, m \in M : \langle q, \langle a, m \rangle, q' \rangle \in \Delta\}.$$

A *state selector function* is a partial function $\phi : Q_R \to Q$ selecting for a non-empty set $P \in Q_R$ an element $p = \phi(P) \in P$. A state of the *left deterministic automaton* $\mathcal{A}_L = \langle \Sigma, Q_L, s_L, Q_L, \delta_L \rangle$ is a pair consisting of a subset of Q and state selector function ϕ. This implies that $Q_L \subseteq 2^Q \times 2^{Q_R \times Q}$ and therefore Q_L is finite. The following induction defines the states and the transition function of the left automaton:

- $s_L := \langle I, \phi_0 \rangle$ where $\phi_0(R) := \begin{cases} \text{any element of } R \cap I & \text{if } R \cap I \neq \emptyset \\ \text{undefined} & \text{otherwise.} \end{cases}$
- For $\langle L, \phi \rangle \in Q_L$ and $a \in \Sigma$ we define $\delta_L(\langle L, \phi \rangle, a) := \langle L', \phi' \rangle$ where
  - $L' := \{q' \mid \exists q \in L, m \in M : \langle q, \langle a, m \rangle, q' \rangle \in \Delta\}$.
  - $\phi'(R') := \begin{cases} \text{any element of } \{q' \in R' \mid \exists m \in M : \langle q, \langle a, m \rangle, q' \rangle \in \Delta\} & \\ \quad \text{if } q = \phi(\delta_R(R', a)) \text{ is defined} & \\ \text{undefined} \quad \text{otherwise.} & \end{cases}$

Given a pair of states $\langle L, \phi \rangle$ and R' of the left and right automaton and $a \in \Sigma$, let $\langle L', \phi' \rangle := \delta_L(\langle L, \phi \rangle, a)$ and $R := \delta_R(R', a)$. Then

$$\psi(\langle L, \phi \rangle, a, R') := \begin{cases} \text{any element of} & \\ \{m \mid \langle \phi(R), \langle a, m \rangle, \phi'(R') \rangle \in \Delta\} & \text{if } \phi(R) \text{ is defined} \\ \text{undefined} & \text{otherwise.} \end{cases}$$

6.3 Pseudo-minimization of monoidal bimachines

When we want to minimize a monoidal bimachine, we cannot minimize the two component automata in a naïve way. The minimization procedure has to take into account the bimachine output function as well.

Definition 6.3.1 Let $\mathcal{B} = \langle \mathcal{M}, \mathcal{A}_L, \mathcal{A}_R, \psi \rangle$ be a monoidal bimachine with component automata $\mathcal{A}_L = \langle \Sigma, L, s_L, L, \delta_L \rangle$ and $\mathcal{A}_R = \langle \Sigma, R, s_R, R, \delta_R \rangle$. The *left profile function* $\psi_L : L \to 2^{\Sigma \times R \times \Sigma^*}$ is defined as

$$\psi_L(q_L) := \{ \langle \sigma, q_R, \psi(q_L, \sigma, q_R) \rangle \mid \sigma \in \Sigma, q_R \in R \}.$$

The *set of left profiles* of $\mathcal{B}$ is defined as $\Gamma_L := \{ \psi_L(q_L) \mid q_L \in L \}$.

Construction of pseudo-minimised bimachine We may think of $\psi_L(q_L)$ as a colour of q_L and consider the coloured automaton $\mathcal{A}_L^c = \langle \Sigma, \Gamma_L, L, s_L, L, \delta_L, \psi_L \rangle$, the set of all state profiles representing the set of colours. The minimization procedure for coloured deterministic finite-state automata described in the second part of Section 3.6 leads to an equivalent minimal coloured deterministic finite-state automaton

$$\mathcal{A}_L^{c\,\prime} = \langle \Sigma, \Gamma_L, \{ [q]_R \mid q \in L \}, [s_L]_R, \{ [q]_R \mid q \in L \}, \delta', col' \rangle$$

where $col'([q]_R) := col(q)$ (see Corollary 3.6.16). We may use

$$\mathcal{A}_L{}' := \langle \Sigma, \Gamma_L, \{ [q]_R \mid q \in L \}, [s_L]_R, \{ [q]_R \mid q \in L \}, \delta' \rangle$$

instead of $\mathcal{A}_L$ as the left automaton and the colouring col' to define the new output function. The equation $col'([q]_R) := col(q)$ directly ensures that the new bimachine has the same output function as $\mathcal{B}$.

Definition 6.3.3 Let $\mathcal{B} = \langle \mathcal{M}, \mathcal{A}_L, \mathcal{A}_R, \psi \rangle$ be a monoidal bimachine. Then the monoidal bimachine $\mathcal{B}' = \langle \mathcal{M}, \mathcal{A}'_L, \mathcal{A}'_R, \psi' \rangle$ constructed by the minimization of $\mathcal{A}_L$ and $\mathcal{A}_R$ (which are treated as coloured automata as described above, with $\psi'([l], \sigma, [r]) := \psi(l, \sigma, r)$) is called the *pseudo-minimal monoidal bimachine equivalent to* $\mathcal{B}$.

6.4 Direct composition of classical bimachines

Formal construction.

Let

$$\begin{aligned} \mathcal{A}_L^N &:= \langle \Sigma, L' \times R' \times L'', \{s'_L\} \times R' \times \{s''_L\}, L' \times R' \times L'', \Delta_L \rangle \\ \mathcal{A}_R^N &:= \langle \Sigma, L' \times R' \times R'', L' \times \{s'_R\} \times \{s''_R\}, L' \times R' \times R'', \Delta_R \rangle \end{aligned}$$

where

1. Δ_L contains all transitions of the form $\langle\langle l'_1, r'_1, l''_1\rangle, a, \langle l'_2, r'_2, l''_2\rangle\rangle$ such that $\delta'_L(l'_1, a) = l'_2$, $\delta'_R(r'_2, a) = r'_1$, $\delta''^*_L(l''_1, \psi'(l'_1, a, r'_2)) = l''_2$,

2. Δ_R contains all transitions of the form $\langle\langle l'_2, r'_2, r''_2\rangle, a, \langle l'_1, r'_1, r''_1\rangle\rangle$ such that $\delta'_L(l'_1, a) = l'_2$, $\delta'_R(r'_2, a) = r'_1$, $\delta''^*_R(r''_2, \rho(\psi'(l'_1, a, r'_2))) = r''_1$.

Let $\mathcal{A}_L$ and $\mathcal{A}_R$ be the deterministic finite state automata constructed from $\mathcal{A}^N_L$ and $\mathcal{A}^N_R$ by the subset construction given in Theorem 3.2.2, where the states are restricted to the ones reachable from the starting states. I.e.

$$\begin{aligned}\mathcal{A}_L &= \langle \Sigma, Q_L, s_L, Q_L, \delta_L\rangle\\ \mathcal{A}_R &= \langle \Sigma, Q_R, s_R, Q_R, \delta_R\rangle\end{aligned}$$

where

$$\begin{aligned}s_L &= \{s'_L\} \times R' \times \{s''_L\}\\ s_R &= L' \times \{s'_R\} \times \{s''_R\}\\ \hat{\delta}_L(A, a) &= \{\langle l'_2, r'_2, l''_2\rangle \mid \exists \langle l'_1, r'_1, l''_1\rangle \in A : \langle\langle l'_1, r'_1, l''_1\rangle, a, \langle l'_2, r'_2, l''_2\rangle\rangle \in \Delta_L\}\\ \hat{\delta}_R(A, a) &= \{\langle l'_1, r'_1, r''_1\rangle \mid \exists \langle l'_2, r'_2, r''_2\rangle \in A : \langle\langle l'_2, r'_2, r''_2\rangle, a, \langle l'_1, r'_1, r''_1\rangle\rangle \in \Delta_R\}\\ Q_L &= \{A \in 2^{L' \times R' \times L''} \mid \exists v \in \Sigma^* : A = \hat{\delta}^*_L(s_L, v)\}\\ Q_R &= \{A \in 2^{L' \times R' \times R''} \mid \exists v \in \Sigma^* : A = \hat{\delta}^*_R(s_R, v)\}\\ \delta_L &= \hat{\delta}_L|_{Q_L \times \Sigma}\\ \delta_R &= \hat{\delta}_R|_{Q_R \times \Sigma}\end{aligned}$$

For states A and B of $\mathcal{A}_L$ and $\mathcal{A}_R$ and $a \in \Sigma$ define

$$\psi(A, a, B) := w \text{ iff}$$

there exist tuples $\langle l'_1, r'_1, l''_1\rangle \in A$ and $\langle l'_2, r'_2, r''_2\rangle \in B$ such that there exist $l''_2 \in L''$ and $r''_1 \in R''$ such that

$$\begin{aligned}\langle\langle l'_1, r'_1, l''_1\rangle, a, \langle l'_2, r'_2, l''_2\rangle\rangle &\in \Delta_L,\\ \langle\langle l'_2, r'_2, r''_2\rangle, a, \langle l'_1, r'_1, r''_1\rangle\rangle &\in \Delta_R,\end{aligned}$$

and $w = \psi''^*(l''_1, \psi'(l'_1, a, r'_2), r''_2)$.

Proposition 6.4.1 *Let $\mathcal{B}'$, $\mathcal{B}''$ be two bimachines over the same alphabet Σ as above, let the bimachine $\mathcal{B}$ and its output function ψ be defined as above. Then ψ is well-defined and $\mathcal{B} := \langle \mathcal{A}_L, \mathcal{A}_R, \psi\rangle$ represents the composition of the bimachines $\mathcal{B}'$ and $\mathcal{B}''$, i.e. $O_{\mathcal{B}} = O_{\mathcal{B}'} \circ O_{\mathcal{B}''}$.*

7 The *C(M)* language

In this chapter we introduce the *C(M)* language. The language is used throughout the rest of the dissertation for implementing and presenting algorithms.

C(M) statements and expressions closely resemble the notation commonly used for the presentation of formal constructions in a Tarskian style set theoretical language. The usual set theoretic objects such as sets, functions, relations, tuples etc. are naturally integrated in the language. In contrast to imperative languages such as C or Java, *C(M)* is a functional declarative programming language. *C(M)* has many similarities with Haskell [Hutton, 2007] but makes use of the standard mathematical notation like SETL [Schwartz et al., 1986]. When implementing the solution to a problem, instead of specifying how to achieve it, we specify the goal itself. In practice, we just formally describe the kind of mathematical object we want to obtain. This allows us to focus on the high-level mathematical steps of a construction as opposed to the low-level implementation details. The *C(M)* compiler translates a well-formed *C(M)* program into efficient C code, which can be executed after compilation. Since it is easy to read *C(M)* programs, a pseudo-code description becomes obsolete. The programs presented below are tested and fully functional and can be compiled without modifications and run on a computer. The compiler for the *C(M)* language is freely available at `http://lml.bas.bg/~stoyan/lmd/C(M).html`.

7.1 Basics and simple examples

Perhaps the best starting point for an introduction to *C(M)* is a short selection of simple examples.

Example 7.1.1 Recall that the composition of two binary relations R_1 and R_2 is defined as $R_1 \circ R_2 := \{\langle a, c\rangle \mid \exists b : \langle a, b\rangle \in R_1, \langle b, c\rangle \in R_2\}$. In *C(M)*, the usual elements for describing sets are available - we may define this composition directly as

$$\text{compose}(R_1, R_2) := \{(a, c) \mid (a, b) \in R_1, (b, c) \in R_2\};$$

Given the two relations R_1 and R_2, the above function returns the set of pairs (a, c) where (a, b) runs over R_1 (b is arbitrary) and (b, c) runs over R_2. Mathematically, the *n-th order composition* $R^{\langle n\rangle}$ of a binary relation R is defined in an inductive manner:

- $R^1 := R$,

- $R^{i+1} := R^i \circ R$

Explicit inductive constructions are a core element of *C(M)*. These definitions start with a base step such as "step 1" where we define a base form of the objects to be constructed. Then "step i+1" explains how to obtain variant $i+1$ of the objects from variant i. An "until" clause explains when the construction is finished. Following this scheme, the n-th order composition may be introduced in *C(M)* as the object "$\mathrm{compose_n}(R, n)$" in the following way:

$\mathrm{compose_n}(R, n) := R'$, **where**
  $R' :=$ **induction**
    step 1 :
      $R'^{(1)} := R$;
    step $i+1$:
      $R'^{(i+1)} := \mathrm{compose}(R'^{(i)}, R)$;
    until $i = n$
    ;
  ;

Mathematically, the transitive closure of a binary relation R is defined as the relation

$$C_R := \bigcup_{i=1}^{\infty} R^i.$$

To construct C_R we may proceed inductively, defining $C_R^{(n)} = \bigcup_{i=1}^{n} R^i$. For a finite relation the inductive step has to be performed until $R^{\langle n+1 \rangle} \subseteq C_R^{(n)}$. In *C(M)* we may use exactly the same procedure and construct C_R using the following induction:

$C_R :=$ **induction**
  step 1 :
    ${C_R}^{(1)} := R$;
  step $n+1$:
    ${C_R}^{(n+1)} := {C_R}^{(n)} \cup \mathrm{compose_n}(R, n+1)$;
  until $\mathrm{compose_n}(R, n+1) \subseteq {C_R}^{(n)}$
  ;

Program 7.1.2 In order to complete this *C(M)* program we have to specify the type of each object. The resulting program has the following form:

1 $\mathcal{REL}$ **is** $2^{\mathbb{N}\times\mathbb{N}}$;

```
2  compose : RELL × RELL → RELL;
3  compose(R_1, R_2) := {(a, c) | (a, b) ∈ R_1, (b, c) ∈ R_2};
4  compose_n : RELL × ℕ → RELL;
5  compose_n(R, n) := R', where
6     R' := induction
7        step 1 :
8           R'^(1) := R;
9        step i + 1 :
10          R'^(i+1) := compose(R'^(i), R);
11       until i = n
12       ;
13    ;
14 transitiveClosure : RELL → RELL;
15 transitiveClosure(R) := C_R, where
16    C_R := induction
17       step 1 :
18          C_R^(1) := R;
19       step n + 1 :
20          C_R^(n+1) := C_R^(n) ∪ compose_n(R, n + 1);
21       until compose_n(R, n + 1) ⊆ C_R^(n)
22       ;
23    ;
```

In mathematics, a fixed object can be described in many distinct ways. Ignoring matters of transparency, it is not important how we specify the object. However, in a computational context the way how we describe an object may have a strong influence on the time needed to compute it.

Program 7.1.3 The following improved construction avoids the two deficiencies. First, it explicitly builds the function F_R, and the composition is performed in the optimized way. And second, the induction is performed by considering at each step one new pair from the set C_R. In this way, starting from $C_R{}^{(0)} := R$, in step $n+1$ we generate the composition of the $n+1$-st element in $C_R^{(n)}$ with the relation R until the set $C_R^{(n)}$ is exhausted.
The program below presents two new features of *C(M)*, the use of a subcase analysis in definitions (Line 9) and the use of the "functionalize"-operator $\mathcal{F}$ in Line 3, cf. Definition 1.1.16.

```
1  transitiveClosure : 2^(ℕ×ℕ) → 2^(ℕ×ℕ);
2  transitiveClosure(R) := C_R, where
3     F_R := F_(1→2)(R);
```

```
4     C_R := induction
5       step 0 :
6         C_R^(0) := R;
7       step n + 1 :
8         (a, b) := (C_R^(n) as (ℕ × ℕ)*)_{n+1};
9         C_R^(n+1) := { C_R^(n) ∪ {(a, c) | c ∈ F_R(b)}   if ! F_R(b)
                       { C_R^(n)                           otherwise;
10      until n = |C_R^(n)|
11      ;
12    ;
```

7.2 Types, terms, and statements in *C(M)*

In this section in the dissertation we describe types, terms and statements.

8 *C(M)* implementation of finite-state devices

In this chapter we present *C(M)* implementations of the main automata constructions. Our aim is to provide full, clear and easy to follow descriptions of the implementations. In some cases the simplicity of the implementation is achieved at the expense of some inefficiency.

8.1 *C(M)* implementations for automata algorithms

Program 8.1.1 We start with the basic definitions and algorithms for finite-state automata. In our formalization, an automaton has a *set* of initial states, and transition labels are arbitrary *words* over the input alphabet.

Program 8.1.2 The following constructions present the regular operations **union**, **concatenation**, and **Kleene star** for finite-state automata.

Program 8.1.3 The next "supplementary constructions" are given for convenience. They can be expressed by our basis functions but help to keep the description of complex automata constructions transparent. We introduce the **Kleene plus** of a given automaton, **optionality** (adding the empty word) for an automaton, the automaton recognizing a given **set of symbols**, and the automaton recognizing **all words** over a given alphabet.

Program 8.1.4 The *C(M)* program for **ε-removal** follows Proposition 2.5.4.

Program 8.1.5 The *C(M)* program for **ε-removal preserving state languages** closely follows Proposition 2.5.6.

Program 8.1.6 The following program for **trimming** an automaton removes all states that are not reachable from any initial state and all states from which no final state can be reached (cf. Definition 2.5.1).

Program 8.1.7 Given an arbitrary finite-state automaton, our next program constructs a **classical** finite-state automaton, i.e., an automaton where each transition label is a word of length ≤ 1.

Program 8.1.8 As in the general case (cf. Program 8.1.6), **trimming** a deterministic automaton means to remove all states that are not reachable from the initial states and all states from which no final state can be reached (Definition 2.5.1).

Program 8.1.9 Below we present a efficient **determinization** construction. Following Theorem 3.2.2 it builds a deterministic automaton where all states are reachable from the new initial state.

Program 8.1.10 The following algorithm (product_Δ) constructs the transition function of an automaton which is the Cartesian product of two input automata.

The above function represents the essential part of the programs for intersection and difference to be described now.

Program 8.1.11 The following *C(M)* program implements **intersection** of deterministic classical finite-state automata as described in Part 1 of Proposition 3.3.2.

Program 8.1.12 The next *C(M)* program implements **difference** of finite-state automata as described in Part 2 of Proposition 3.3.2.

Program 8.1.13 The following *C(M)* program implements **reversal** of finite-state automata as described in Proposition 3.3.3.

Program 8.1.14 The algorithm implements a **minimization** procedure for deterministic finite-state automata based on the inductive construction presented in Corollary 3.5.4, using the functions defined in Proposition 3.5.6.

Program 8.1.15 Below we implement two basic functions on deterministic finite-state automata. The first function 'C_δ' implements the transitive closure of the transition function δ^*. The second function 'stateseq' returns the sequence of states on the automaton path starting from the given input state that is labeled with the given word.

Example 8.1.16 In order to illustrate the use of the algorithms presented in this section we provide a program for constructing a deterministic finite-state automaton that recognizes all valid dates of the Gregorian calendar formated as expressions of the form

```
AUGUST 11, 1996
```

We loosely follow the approach in [Karttunen et al., 1997a]. Date expressions like "`FEBRUARY 30, 2015`" or "`APRIL 31, 1921`" are easily described as incorrect. More challenging is the case with leap days. "`FEBRUARY 29, 2000`" and "`FEBRUARY 29, 2016`" are valid dates but "`FEBRUARY 29, 2017`" and "`FEBRUARY 29, 1900`" do not exist.

8.2 *C(M)* programs for classical finite-state transducers

In this section we present *C(M)* implementations for the main algorithms for n-tape automata as described in Secton 4.1. We only look at 2-tape automata over the free monoid, i.e., classical finite-state transducers.

Program 8.2.1 As a first step the types used for finite-state transducers over the free monoid are introduced, and the function for **renaming** the states of a classical finite-state transducer is defined. We then introduce a transducer for translating a given word into a second word.

Program 8.2.2 The next program describes **union**, **concatenation**, **Kleene star**, **Kleene plus**, **optionality**, and $\langle\varepsilon,\varepsilon\rangle$**-removal** for finite-state transducers.

Program 8.2.3 The program below takes two 1-tape automata as input and computes a transducer representing the **Cartesian product** of the two input automaton languages as in Proposition 4.2.1, Part 1.

Program 8.2.4 Following Proposition 4.2.1 the following program presents constructions for **projections** of a 2-tape automaton on the first and second tape, **inverse relation** and the **identity** relation for a given automaton language.

Program 8.2.5 The next algorithm, given a finite-state transducer, constructs an equivalent **classical** 2-tape letter automaton, which means that all transition labels are in $\Sigma^\varepsilon \times \Sigma^\varepsilon$. Following Proposition 4.3.3 we expand the transitions containing labels with words with more than one symbol by introducing new intermediate states.

Program 8.2.6 The following algorithm constructs the finite-state transducer that represents the **composition** of two finite-state transducers according to Proposition 4.3.4.

Program 8.2.7 The next algorithm constructs the finite-state transducer that represents the **reversal** of a finite-state transducer.

Real-time translation and pseudo-determinization of transducers

Program 8.2.8 The following algorithms implement the removal of transducer transitions with label ε on the upper tape as described in the proof of Proposition 4.4.8, and the conversion of a transducer to a **real-time transducer**.

Program 8.2.9 The next algorithm constructs a **pseudo-deterministic transducer** equivalent to a given input transducer using the steps described in the proof of Proposition 3.7.2.

Program 8.2.10 The following algorithm constructs a **pseudo-minimal transducer** following Proposition 3.7.4.

Deciding functionality of transducers

Program 8.2.11 The next program constructs the **squared output transducer** for a real-time transducer in accordance with Definition 4.6.4.

Program 8.2.12 This program decides the **functionality** of a transducer following Corollary 4.6.7, Proposition 4.6.8 and Corollary 4.6.11.

Example 8.2.13 The following program realizes the functionality of a fully-fledged spell checker. Similar implementations are used e.g. in [Ringlstetter et al., 2007, Mitankin et al., 2014]. It implements a function testing for an input word w if w is in a given background dictionary, also retrieving the set of dictionary words that are "close" to the given word. This set represents the correction candidates for the given word. Closeness is defined in terms of the Levenshtein distance. We recall that the Levenshtein distance between two words is the minimal number of symbol substitutions, deletions and insertions required for transforming the first word into the second one. This code can be applied on any suitable word list.

Remark 8.2.14 The above functions can be applied for large dictionaries and provide a practical solution to spell checking and other similarity based computations. Nevertheless some steps can be substantially improved.

1. The minimal deterministic finite-state automaton for the dictionary can be constructed in a much more efficient way [Daciuk et al., 2000].

2. The deterministic Levenshtein automaton can be constructed directly using the method in [Schulz and Mihov, 2002]. A more sophisticated approach presented in [Mihov and Schulz, 2004] constructs the *universal* deterministic Levenshtein automaton which does not depend on the input word. A comprehensive study of universal Levenshtein finite-state automata and transducers is given in [Mitankin et al., 2011].

3. The words in the intersection of the dictionary automaton and the Levenshtein automaton can be obtain more efficiently with a parallel traversal procedure, avoiding the construction of the intersection automaton.

8.3 *C(M)* programs for deterministic transducers

In this section we present the algorithms for construction and minimization of deterministic transducers.

Program 8.3.1 The following algorithm constructs a **subsequential finite-state transducer** from a finite-state transducer with the bounded variation property, closely following the inductive construction presented in Section 5.2.

Program 8.3.2 The next program tests the **bounded variation poperty** for a transducer in accordance with Theorem 5.3.4 and Lemma 5.3.5.

Minimization of subsequential transducers

Program 8.3.3 The program below defines the type for **subsequential finite-state transducers with initial output** and the conversion procedure from and to ordinary subsequential finite-state transducer according to Definition 5.4.4.

Program 8.3.4 This algorithm returns the **expanded output automaton** for a given subsequential transducer as defined in Definition 5.5.11.

Program 8.3.5 The function below calculates the **maximal state output function** *mso* for a given transducer $\mathcal{T}$ according to Corollary 5.5.15.

Program 8.3.6 The following program converts a subsequential transducer into **canonical form** as defined in Definition 5.5.2.

Program 8.3.7 The next program presents the **pseudo-minimization** and **minimization** procedures for subsequential transducers.

Program 8.3.8 The following function C_λ realizes the generalized transition output function λ^* of a subsequential transducer, given the transition function δ and the transition output function λ.

Program 8.3.9 The following algorithm constructs a subsequential transducer that represents the **composition** of two subsequential transducers following Proposition 5.1.10.

Phonetization of numbers as an application In order to demonstrate the use of the above constructions we present a program for constructing a minimal subsequential transducer that maps a number written as a sequence of digits to its phonetization. This kind of functionality is needed, e.g., for speech synthesis.

Example 8.3.10 For the current implementation we use the phoneme set of the Carnegie Mellon University pronouncing dictionary[1]. In our example we limit the input range to numbers between 1 and 999999. As a matter of fact this segment could be easily extended.

Remark 8.3.11 The minimal subsequential transducer for a finite function can be constructed much more efficiently applying the algorithm presented in [Mihov and Maurel, 2001].

8.4 *C(M)* programs for bimachines

In this section we present the algorithms for construction, composition and normalization of bimachines.

Program 8.4.1 The following algorithm converts a **functional transducer into a bimachine** following the construction given in the proof of Proposition 6.2.5.

[1] `http://www.speech.cs.cmu.edu/cgi-bin/cmudict`

Program 8.4.2 The next algorithm constructs a **pseudo-minimal bimachine** in accordance with Definition 6.3.3 by minimizing the left and the right automaton of the bimachine (considered as coloured automata using state profiles as colours) following the procedure described in Section 6.3.

Program 8.4.3 The algorithm computes the **composition of two bimachines** following the construction presented in Section 6.4.

Bignum arithmetics with bimachines as an application We complete the section with an example presenting the use of bimachines for implementing basic arithmetic operations on unbounded natural numbers.

Example 8.4.4 Any natural number (e.g. 5389) can be represented as string of digits such as `['5','3','8','9']`. The programs defined below take as input an arbitrary natural number $K \in \mathbb{N}$. Given the number K an "addition" bimachine is computed that reads as input a second natural number x represented as a string of digits as indicated above. The output of the addition bimachine is the representation of $x + K$ as a string of digits. In a similar way, for a given input number K, bimachines for the operations $x \mapsto x - K$ (subtraction, defined for inputs $x \geq K$), $x \mapsto x \cdot K$ (multiplication), $x \mapsto x/K$ (division), and for the remainder operation after dividing by K are computed. In each case, input and output numbers are represented as strings.

Author's Contributions

The main scientific contributions of this dissertation are:

1. A complete and coherent presentation of the theory of finite-state automata, transducers and bimachines, together with detailed proofs of the main properties and correctness of constructions, is given, which combines abstract algebraic terms with computationally efficient constructions.

2. A decision procedure for deciding the bounded variation property of a finite-state transducers has been developed, which can be integrated in the sequentialization construction. In previous approaches (see, for example, [Roche and Schabes, 1997a]), in order to avoid the endless operation of the sequentialization construction, the bounded variation property has to be tested in advance by a complex special algorithm. With the presented method, this problem is solved in an elegant way

by adding one additional check within the construction (see Theorem 5.3.8).

3. A new construction with polynomial complexity for canonization of a subsequential transducer is presented (see Corollary 5.5.15 and Remark 5.5.16). The advantage of the new construction is its good efficiency and the use of a fully automata-based approach.

4. A new construction has been developed for constructing a bimachine from a finite-state transducer (see Proposition 6.2.5). The advantage of the new construction is the avoidance of the pre-construction for obtaining an unambiguous finite-state transducer. For certain classes of transducers, a variant of the new construction results in an exponentially smaller number of states of the derived bimachine [Gerdjikov et al., 2017].

5. A construction together with correctness proof was obtained for direct composition of bimachines (Section 6.4). Unlike the standard approach, which requires the conversion of the bimachines to letter finite-state transducers and vice versa, in the new construction the resulting bimachine is constructed directly.

The main scientific-applicational contributions of the presented dissertation are:

1. A new programming language $C(M)$ has been developed that allows us to focus on abstract-level mathematical steps in describing algorithms instead of describing low-level execution details.

2. Working implementations in $C(M)$ of all major constructions for finite-state automata, transducers and bimachines are presented.

3. The dissertation provides implementations of real-life programs based on finite-state automata, transducers and bimachines for a number of practical tasks such as spelling correction, phonetization, bignum arithmetic, and more.

References

[Allauzen et al., 2007] Allauzen, C., Riley, M., Schalkwyk, J., Skut, W., and Mohri, M. (2007). Openfst: A general and efficient weighted finite-state transducer library. In Holub, J. and Žďárek, J., editors, *Implementation*

and Application of Automata, pages 11–23, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Angelova and Mihov, 2008] Angelova, G. and Mihov, S. (2008). Finite state automata and simple conceptual graphs with binary conceptual relations. In *Supplementary Proceedings of the 16th International Conference on Conceptual Structures, ICCS 2008, Toulouse, France, July 7-11, 2008*, pages 139–148.

[Beesley and Karttunen, 2003] Beesley, K. and Karttunen, L. (2003). *Finite State Morphology*. CSLI studies in computational linguistics: Center for the Study of Language and Information. CSLI Publications.

[Berstel, 1979] Berstel, J. (1979). *Transductions and context-free languages*. Leitfäden der angewandten Mathematik und Mechanik. Teubner.

[Daciuk et al., 2000] Daciuk, J., Mihov, S., Watson, B., and Watson, R. (2000). Incremental construction of minimal acyclic finite state automata. *Computational Linguistics*, 26(1):3–16.

[Eilenberg, 1974] Eilenberg, S. (1974). *Automata, Languages, and Machines - Volume A*, volume volume 59 of Pure and Applied Mathematics. Academic Press.

[Eilenberg, 1976] Eilenberg, S. (1976). *Automata, Languages, and Machines - Volume B*. Academic Press.

[Ganchev et al., 2008] Ganchev, H., Mihov, S., and Schulz, K. U. (2008). One-letter automata: How to reduce k tapes to one. In Hamm, F and Kepser, S, editor, *LOGICS FOR LINGUISTIC STRUCTURES*, volume 201 of *Trends in Linguistics-Studies and Monographs*, pages 35–55. WALTER DE GRUYTER GMBH. Conference in Honor of Uwe Monnich on his 70th Birthday, Freudenstadt, GERMANY, NOV, 2004.

[Gerdemann and van Noord, 1999] Gerdemann, D. and van Noord, G. (1999). Transducers from rewrite rules with backreferences. In *Proceedings of the 9th Conference of the European Chapter of the Association for Computational Linguistics (EACL 99)*, pages 126–133.

[Gerdjikov and Mihov, 2017a] Gerdjikov, S. and Mihov, S. (2017a). Myhill-nerode relation for sequentiable structures. *CoRR*, abs/1706.02910.

[Gerdjikov and Mihov, 2017b] Gerdjikov, S. and Mihov, S. (2017b). Over which monoids is the transducer determinization procedure applicable?

In *Language and Automata Theory and Applications - 11th International Conference, LATA 2017, Umeå, Sweden, March 6-9, 2017, Proceedings*, volume 10168 LNCS, pages 380–392.

[Gerdjikov et al., 2017] Gerdjikov, S., Mihov, S., and Schulz, K. U. (2017). A simple method for building bimachines from functional finite-state transducers. In Carayol, A. and Nicaud, C., editors, *Implementation and Application of Automata*, volume 10329 LNCS, pages 113–125. Springer International Publishing.

[Hopcroft et al., 2006] Hopcroft, J. E., Motwani, R., and Ullman, J. D. (2006). *Introduction to Automata Theory, Languages, and Computation*. Pearson. 3nd edition.

[Hulden, 2009] Hulden, M. (2009). *Finite-State Machine Construction Methods and Algorithms for Phonology and Morphology*. PhD thesis, University of Arizona.

[Hutton, 2007] Hutton, G. (2007). *Programming in Haskell*. Cambridge University Press.

[Kaplan and Kay, 1994] Kaplan, R. and Kay, M. (1994). Regular models of phonological rule systems. *Computational Linguistics*, 20(3):331–279.

[Karttunen, 1997] Karttunen, L. (1997). The replace operator. In Roche, E. and Schabes, Y., editors, *Finite-State Language Processing*, pages 117–147. MIT Press.

[Karttunen et al., 1997a] Karttunen, L., Chanod, J.-P., Grefenstette, G., and Schiller, A. (1997a). Regular expressions for language engineering. *Journal of Natural Language Engineering*, 2(4):307–330.

[Karttunen et al., 1997b] Karttunen, L., Gaál, T., and Kempe, A. (1997b). Xerox finite-state tool. Technical report, Xerox Corporation.

[Kozen, 1997] Kozen, D. C. (1997). *Automata and Computability*. Springer, New York, Berlin.

[Lewis and Papadimitriou, 1998] Lewis, H. R. and Papadimitriou, C. H. (1998). *Elements of the Theory of Computation*. Prentice-Hall, Upper Saddle River, New Jersey. 2nd edition.

[Maurel and Guenthner, 2005] Maurel, D. and Guenthner, F. (2005). *Automata and Dictionaries*. Texts in Computer Science. College Publications.

[Mihov and Maurel, 2001] Mihov, S. and Maurel, D. (2001). Direct construction of minimal acyclic subsequential transducers. In *Proceedings of the Conference on Implementation and Application of Automata CIAA'2000*, volume 2088 of *LNCS*, pages 217–229. Springer.

[Mihov and Schulz, 2019] Mihov, S. and Schulz, K. (2019). *Finite-State Techniques: Automata, Transducers and Bimachines*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press.

[Mihov and Schulz, 2004] Mihov, S. and Schulz, K. U. (2004). Fast approximate search in large dictionaries. *Computational Linguistics*, 30(4):451–477.

[Mitankin et al., 2014] Mitankin, P., Gerdjikov, S., and Mihov, S. (2014). An approach to unsupervised historical text normalisation. In *Digital Access to Textual Cultural Heritage 2014, DATeCH 2014, Madrid, Spain, May 19-20, 2014*, pages 29–34.

[Mitankin et al., 2011] Mitankin, P., Mihov, S., and Schulz, K. U. (2011). Deciding word neighborhood with universal neighborhood automata. *Theoretical Computer Science*, 412(22):2340–2355.

[Mohri, 1996] Mohri, M. (1996). On some applications of finite-state automata theory to natural language processing. *Journal of Natural Language Engineering*, 2:1–20.

[Mohri, 1997] Mohri, M. (1997). Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(2):269–311.

[Mohri, 2000] Mohri, M. (2000). Minimization algorithms for sequential transducers. *Theoretical Computer Science*, 234:177–201.

[Mohri et al., 2008] Mohri, M., Pereira, F., and Riley, M. (2008). Speech recognition with weighted finite-state transducers. In *Springer Handbook of Speech Processing*, pages 559–584. Springer.

[Mohri and Sproat, 1996] Mohri, M. and Sproat, R. (1996). An efficient compiler for weighted rewrite rules. In *Proceedings of the 34th Meeting of the Association for Computational Linguistics (ACL'96)*, pages 231–238, Santa Cruz, CA.

[Navarro and Raffinot, 2002] Navarro, G. and Raffinot, M. (2002). *Flexible Pattern Matching in Strings*. Cambridge University Press, Cambridge, UK.

[Reutenauer and Schützenberger, 1991] Reutenauer, C. and Schützenberger, M. P. (1991). Minimization of rational word functions. *SIAM J. Computing*, 20(4):669–685.

[Ringlstetter et al., 2007] Ringlstetter, C., Schulz, K. U., and Mihov, S. (2007). Adaptive text correction with web-crawled domain-dependent dictionaries. *ACM Transactions on Speech and Language Processing*, 4(4).

[Roche and Schabes, 1997a] Roche, E. and Schabes, Y. (1997a). Deterministic part-of-speech tagging with finite-state transducers. In Roche, E. and Schabes, Y., editors, *Finite-State Language Processing*, Language, Speech, and Communication, pages 205–240. The MIT Press.

[Roche and Schabes, 1997b] Roche, E. and Schabes, Y. (1997b). Introduction. In Roche, E. and Schabes, Y., editors, *Finite-State Language Processing*, pages 1–66. MIT Press.

[Sakarovitch, 2009] Sakarovitch, J. (2009). *Elements of Automata Theory*. Cambridge University Press, New York, NY, USA.

[Schmid, 2006] Schmid, H. (2006). A programming language for finite state transducers. In Yli-Jyrä, A., Karttunen, L., and Karhumäki, J., editors, *Finite-State Methods and Natural Language Processing*, pages 308–309, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Schulz and Mihov, 2002] Schulz, K. U. and Mihov, S. (2002). Fast String Correction with Levenshtein-Automata. *International Journal of Document Analysis and Recognition*, 5(1):67–85.

[Schützenberger, 1961] Schützenberger, M.-P. (1961). A remark on finite transducers. *Information and Control*, 4:185–196.

[Schwartz et al., 1986] Schwartz, J. T., Dewar, R. B., Schonberg, E., and Dubinsky, E. (1986). *Programming with Sets; an Introduction to SETL*. Springer-Verlag, Berlin, Heidelberg.