University of Niš
Faculty of Sciences and Mathematics
Department of Computer Science

**Zorana Z. Jančić**

# ALGORITHMS FOR DETERMINIZATION OF WEIGHTED AND FUZZY AUTOMATA

PhD thesis

Niš, 2014

University of Niš
Faculty of Sciences and Mathematics
Department of Computer Science

**Zorana Z. Jančić**

# ALGORITHMS FOR DETERMINIZATION OF WEIGHTED AND FUZZY AUTOMATA

PhD thesis

Niš,2014

Univerzitet u Nišu
Prirodno-matematički fakultet
Departman za računarske nauke

**Zorana Z. Jančić**

# ALGORITMI ZA DETERMINIZACIJU TEŽINSKIH I FAZI AUTOMATA

Doktorska disertacija

Niš,2014

**Supervisor:** **Jelena Ignjatović**

Associate Professor

Faculty of Sciences and Mathematics

University of Niš,Serbia

**Members of the Commission:** **Miroslav Ćirić**

Full Professor

Faculty of Sciences and Mathematics

University of Niš, Serbia

**Heiko Vogler**

Full Professor

Faculty of Computer Science

TechnischeUniversität,Dresden,Germany

**Manfred Droste**

Full Professor

Institut für Informatik

Universität Leipzig, Germany

**Aleksandar Stamenković**

Assistant Professor

Faculty of Sciences and Mathematics

University of Niš, Serbia

**Date of Defense**

# Contents


**1 Fundamental concepts** . . . . . 5
1.1. Strong bimonoids and semirings . . . . . 5
1.2. Ordered sets and lattices . . . . . 9
1.3. Complete residuated lattices . . . . . 12
1.4. Fuzzy sets and fuzzy relations . . . . . 17

**2 Weighted and fuzzy automata** . . . . . 23
2.1. Weighted finite automata and series over strong bimonoids . . 23
2.2. Fuzzy automata . . . . . 25
2.3. Crisp-deterministic fuzzy (weighted) automata . . . . . 27
2.4. Afterset and factor fuzzy automata . . . . . 28

**3 Reduced Nerode automaton** . . . . . 31
3.1. Nerode automaton . . . . . 33
3.2. Definition of the reduced Nerode automaton . . . . . 34
3.3. Finiteness conditions . . . . . 36
3.4. Algorithm for computing the reduced Nerode automaton . . . . 40

**4 Brzozowski type determinization for fuzzy automata** . . . . . 45
4.1. Reverse Nerode automaton . . . . . 46
4.2. Fuzzy languages associated with states . . . . . 47
4.3. Brzozowski automaton . . . . . 50
4.4. Algorithm for computing the Brzozowski automaton . . . . . 51

**5 Simultaneous determinization and state reduction** . . . . . 55
5.1. Right and left invariant fuzzy relations . . . . . 57
5.2. Two-in-one: determinization and state reduction . . . . . 58
5.3. Two-in-one algorithms . . . . . 66
5.4. Computational examples . . . . . 72

**6 Determinization by means of the degree of language inclusion** . . 83
6.1. The degree of language inclusion . . . . . 84
6.2. The automaton constructed by means of the degree of language inclusion . . . . . 85
6.3. Algorithm and a computational example . . . . . 88

**A C codes** . . . . . 93

**B Algoritmi za determinizaciju težinskih i fazi automata** . . . . . 113

**C Biography** . . . . . 135

**References** . . . . . 137
. . . . . 137

# Preface

Determinization of nondeterministic finite automata is one of the most elaborated problems in the automata theory. It is the procedure of transforming a given nondeterministic automaton into a language equivalent deterministic automaton, which plays a significant role in many areas of computer science, such as natural language processing, system verification and testing, etc., as well as in many fields outside of computer science, such as molecular biology. Numerous algorithms have been developed for determinization of nondeterministic automata, and the standard one is the *subset construction*. This algorithm converts a nondeterministic automaton with $n$ states to an equivalent deterministic automaton with up to $2^n$ states. In the worst case, subset construction yields a deterministic automaton that is exponentially larger than the input nondeterministic automaton, which sometimes makes the construction impractical for large nondeterministic automata. However, this determinization algorithm is renowned for its good performance in practice. Recently, several algorithms based on the subset construction, that are more memory efficient and produce smaller deterministic automata than the subset construction, have been developed. Some of these algorithms even produce a minimal deterministic automaton equivalent to a given nondeterministic automaton. One of the most important algorithms for determinization of nondeterministic automata is the Brzozowski's determinization algorithm. Given a nondeterministic automaton, this algorithm produces a minimal language equivalent deterministic automaton directly. Brzozowski's algorithm is based on successive application of two simple operations, the reversion and accessible subset construction, twice. Despite its worst-case exponential time complexity, the algorithm has recently gained popularity due to its excellent performance in practice, where it outperforms theoretically faster algorithms.

Weighted and fuzzy automata are classical nondeterministic automata in which transitions, initial and final states take values from certain structures. For weighted automata these values are called weights, and are usually taken from semirings, and for fuzzy automata they are called truth values, and are

taken from certain ordered structures, the most often from lattice-ordered structures. These values may be used for modeling the probability or possibility of the successful transition, the amount of energy needed for successful transition, the cost paid when entering a certain state and etc. Determinization of a weighted/fuzzy automaton is considered here as the procedure of its conversion into an equivalent crisp-deterministic weighted/fuzzy automaton. This kind of determinization was first studied by Bělohlávek [12], for fuzzy finite automata over complete lattices and Li and Pedrycz [72], for fuzzy finite automata over lattice ordered monoids. The algorithms provided there generalize the subset construction. Another algorithm for determinization of fuzzy automata over complete residuated lattices, which is also a generalization of the subset construction, was given by Ignjatović et al. [54]. For any input this algorithm generates a smaller crisp-detreministic automaton than the algorithms developed by Bělohlávek and Li i Pedrycz. Since this crisp-deterministic fuzzy automaton can be alternatively constructed by means of the Nerode right congruence of the original fuzzy finite automaton, it was called the *Nerode automaton* of this fuzzy finite automaton. The Nerode automaton was originaly constructed for fuzzy finite automata over a complete residuated lattice, but it was noted that the same construction can be also applied to fuzzy finite automata over a lattice-ordered monoid, and moreover, to weighted finite automata over a semiring.

Determinization of weighted finite automata over semirings was also studied in [3, 65, 81, 82], but it was based on a different concept of determinism. Namely, a weighted finite automaton was called there deterministic or subsequential if it has a unique initial state and if there is at most one transition with a given input letter out of each state.

In recent times, many authors studied weighted automata with weights in more general structures, called strong bimonoids, which can be viewed as semirings that might lack distributivity. Examples of strong bimonoids include the real unit interval [0,1] with t-conorm and t-norm from multivalued logic, the string bimonoid of all words over an alphabet arising in natural language processing, the algebraic cost structure from algebraic path problems, etc. Important natural examples of strong bimonoids are ortho-modular lattices, which serve as a basis of quantum logics where distributivity typically fails. The study of weighted automata based on quantum logic, i.e. weighted automata with weights in ortho-modular lattices, has recently gained big popularity. Recently, Droste et al. [37] have initiated a study of weighted finite automata over arbitrary strong bimonoids, and Ćirić et al. [27] developed algorithms and methods for determinization of these automata. Due to the lack of distributivity, there are three different definitions of the behavior of weighted finite automata over strong bimonoids: the run semantics, the initial algebra semantics, and the transition semantics. Nerode automaton was also constructed for weighted automata over strong bimonoid, and it was proved that it is equivalent to the original automaton with respect to the initial algebra semantics. The Myhill automaton and the run automaton are

crisp-deterministic weighted automata equivalent to the original weighted automaton with respect to transition and run semantics respectively. In general, the cardinalities of these tree crisp-deterministic weighted automata are incomparable. However, in the case of semirings all three semantics coincide for every weighted finite automaton, and in this case the Nerode automaton has always smaller cardinality than the corresponding Myhill and run automaton.

The main aim of this dissertation is to develop new methods and algorithms for determinization of weighted and fuzzy finite automata, some of which are canonization methods, the methods which simultaneously perform determinization and minimization.

In the first chapter, basic concepts of the lattice theory will be considered. Special attention will be aimed to complete residuated lattices, which are used as the underlying truth structure in the fuzzy setting. Further, the notion of fuzzy sets and fuzzy relations will be introduced. Afterwards, the main notions from the semiring theory, as well as the notions of strong bimonoids, vectors and matrices over semirings, and formal power series will be defined.

In the second chapter we present some basic notions and results concerning weighted automata and fuzzy automata and languages. In accordance with tree different kinds of behavior of weighted automata over a strong bimonoid, the run semantics, initial algebra semantics and transition semantics will be observed. Moreover, the notions of crisp-deterministic weighted and fuzzy automata will be given. We also introduce the concepts of factor fuzzy and weighted automata with respect to congruence relations on their sets of states, as well as homomorphisms for crisp-deterministic weighted and fuzzy automata.

In Chapter 3, for a given weighted finite automaton over a strong bimonoid, we provide the construction of its *reduced Nerode automaton*, which is crisp-deterministic and equivalent to the original weighted automaton with respect to the initial algebra semantics. In addition, it will be shown that the reduced Nerode automaton is even smaller than the Nerode automaton, which was previously used in determinization related to this semantics. The necessary and sufficient conditions under which the reduced Nerode automaton is finite will be determined and an efficient algorithm which computes the reduced Nerode automaton whenever it is finite, will be provided. In determinization of weighted finite automata over semirings and fuzzy finite automata over lattice-ordered monoids this algorithm gives smaller crisp-deterministic automata than any other known determinization algorithm.

In Chapter 4 the well-known Brzozowski's determinization method for nondeterministic automata is adapted to the fuzzy framework. This method gives better results than all previously known methods for determinization of fuzzy automata developed by Bělohlávek [12], Li and Pedrycz [72], Ignjatović et al. [54], and Jančić et al. [61]. Namely, as in the case of ordinary nondeterministic automata, Brzozowski type determinization of a fuzzy automaton results in a minimal crisp-deterministic fuzzy automaton

equivalent to the starting fuzzy automaton, and it will be shown that there are cases when all previous methods result in infinite automata, while Brzozowski type determinization results in a finite one. Here we deal with fuzzy automata over complete residuated lattices, but identical results can also be obtained in a more general context, for fuzzy automata over lattice-ordered monoids, and even for weighted automata over commutative semirings.

In the fifth chapter we provide two-in-one algorithms that simultaneously perform determinization and state reduction. The idea of combining determinization and state reduction methods emerged from the need to find such methods which will mitigate the potential enormous growth of the number of states during the determinization. The algorithms developed here perform better than all previous determinization algorithms for fuzzy finite automata, in the sense that they produce smaller automata, while require the same computation time. The only exception is the above mentioned Brzozowski type determinization algorithm, which produces a minimal crisp-deterministic fuzzy automaton, but we will see that the algorithms created here can also be used within the Brzozowski type algorithm and improve its performances.

In Chapter 6 we propose another canonization algorithm for fuzzy automata over complete residuated lattices. The motivation for this algorithm came from the papers by van Glabbeek and Ploeger [120, 121], who have dealt with nondeterministic automata, but here we use a completely different approach and methodology. This approach is based on the inclusion degrees for particular fuzzy languages associated with a given fuzzy finite automaton. The new construction of the minimal crisp-deterministic fuzzy automaton of a given fuzzy finite automaton, which we present here, is generally different from the Brzozowski's construction, but like the Brzozowski's construction includes the construction of the reverse Nerode automaton as its first step.

Finally, I would like to express my gratitude to my mentor, Professor Jelena Ignjatović, for her outstanding contribution to the preparation of this dissertation, for her support and her confidence. Beyond, I would like to thank Professor Miroslav Ćirić, for the generous help, friendly support and encouragement in scientific research. Next, I want to thank Mr.Ivan Stanković for helping designing the program which implements the results from the dissertation.

Last, but not the least I want to thank my family and friends, who have supported me and supplied the necessary understanding during the preparation of the thesis. Thank you for keeping my spirits up.

# Chapter 1
# Fundamental concepts

In this chapter we explain the basic algebraic concepts that will be used throughout this thesis.

In Section 1.1. we consider strong bimonoids and semirings, some properties of these structures that are important for our research, and we list their most commonly used particular types. Further, we recall the definition of the vectors and matrices over arbitrary strong bimonid and products between them. In Section 1.2. we deal with binary relations on an arbitrary set, properties of binary relations as well as ordered sets and lattices.

Section 1.3. is dedicated to residuated lattices, which are used as the underlying truth structure in the fuzzy setting. In the last section, we consider fuzzy sets and relations, together with fuzzy equivalences and fuzzy quasi orders.

## 1.1. Strong bimonoids and semirings

In this section, we consider bimonoids and semirings.

A *monoid* is a triple $(M, \cdot, 1)$, where

(1) $M$ is a non-empty set,
(2) a binary operation $\cdot$ on $M$, called the *multiplication*, is associative,
(3) an element $1 \in M$, called a *neutral element*, satisfies

$$m \cdot 1 = 1 \cdot m = m, \qquad \text{for every } m \in M.$$

If the multiplication operation and the neutral element of $M$ are understood, then we denote the monoid simply by $M$. If the operation $\cdot$ is commutative, that is, if $m \cdot n = n \cdot m$ for every $m, n \in M$, the monoid $M$ is called *commutative*. A commutative monoid $M$ is often denoted by $(M, +, 0)$. For example, if we denote by $R$ the set of all real numbers and by $N$ the set of all natural numbers, than we can easily draw the conclusion that $(R, +, 0)$ and $(N, \cdot, 1)$ are com-

mutative monoids. One of the most important types of a monoids is the *free monoid* $X^*$, generated by a nonempty set $X$, called an *alphabet*. The elements of $X$ are called *letters*. The set $X^*$ is composed of all the *words* over $X$:

$$X^* = \{x_1...x_n \,|\, x_1,...,x_n \in X, n \geqslant 1\} \cup \{\varepsilon\}.$$

The multiplication operation is the *concatenation* and the neutral element of $X^*$ is the *empty word* $\varepsilon$. The length of a nonempty word $w = x_1...x_n$, denoted by $|w|$, is defined to be $n$, and $|\varepsilon| = 0$.

A *bimonoid* is a structure $(K, +, \cdot, 0, 1)$ where:

(1) $(K, +, 0)$ is a monoid with the neutral element 0;
(2) $(K, \cdot, 1)$ is a monoid with the neutral element 1.

As usual, we identify the structure $(K, +, \cdot, 0, 1)$ with its carrier set $K$.

A *strong bimonoid* is a structure $(K, +, \cdot, 0, 1)$ where:

(1) $(K, +, \cdot, 0, 1)$ is a bimonoid;
(2) the operation + is commutative;
(3) 0 acts as multiplicative zero, i.e., $k \cdot 0 = 0 = 0 \cdot k$ for every $k \in K$.

We say that a strong bimonoid $K$ is *right distributive*, if it satisfies

$$(a + b) \cdot c = a \cdot c + b \cdot c,$$

for every $a, b, c \in K$, and we call $K$ *left distributive*, if

$$a \cdot (b + c) = a \cdot b + a \cdot c,$$

for every $a, b, c \in K$.

*Example 1.1.* The set $N$ of natural numbers together with $\infty$ forms a strong bimonoid called the *tropical bimonoid* $(N \cup \{\infty\}, +, \min, 0, \infty)$, where the addition operation is the classical addition of natural numbers, whereas the multiplication operation is defined to be the minimum. The neutral element for addition is 0 and the natural element for multiplication is $\infty$.

*Example 1.2.* The real unit interval $[0, 1]$, with the ordinary multiplication $\cdot$ of reals as the addition operation, and the multiplication operation defined by

$$a \oplus b = a + b - a \cdot b \qquad (\textit{algebraic sum})$$

is a strong bimonoid with neutral elements 0 and 1, respectively.

*Example 1.3.* The real unit interval $[0, 1]$, with the ordinary multiplication $\cdot$ of reals as the addition operation, and the multiplication operation defined by

$$a \oplus b = min(a + b, 1) \qquad (\textit{bounded sum})$$

is a strong bimonoid with neutral elements 0 and 1, respectively.

*Example 1.4.* Any bounded lattice $L$ together with $\wedge$ and $\vee$ operations as the addition and multiplication operations, respectively, is a strong bimonoid.

*Example 1.5.* For any alphabet $X$, the set $X^* \cup \{\infty\}$ of all the words over $X$ together with $\infty$, forms a strong bimonoid $(X^* \cup \{\infty\}, \wedge, \cdot, \infty, \varepsilon)$, called the *string bimonoid*, where

$\wedge$ is the longest common prefix operation;

$\cdot$ is usual concatenation of words;

$w \wedge \infty = \infty \wedge w = w, \quad w \cdot \infty = \infty \cdot w = \infty, \quad$ for every $w \in X^* \cup \{\infty\}$.

A *semiring* is a set $S$ together with two binary operations $+$ and $\cdot$ and two constant elements 0 and 1 such that:

(1) $(S, +, 0)$ is a commutative monoid,
(2) $(S, \cdot, 1)$ is a monoid,
(3) the distributivity laws hold for every $a, b, c \in S$,
(4) $0 \cdot a = a \cdot 0 = 0$ for every $a \in S$.

In other words, a semiring is a strong bimonoid which is right and left distributive. A semiring $S$ is called commutative if $a \cdot b = b \cdot a$ for every $a, b \in S$.

Here we give some of the most important examples of semirings:

*Example 1.6.* The set of all natural numbers $\mathbb{N}$ together with $\infty$ forms a semiring with respect to minimum and addition operation. This semiring, called the *tropical semiring*, has $\infty$ and 0 as neutral elements. The set of all positive real numbers $\mathbb{R}_+$ together with $\infty$ forms a semiring with respect to the same operations.

*Example 1.7.* The set of all natural numbers $\mathbb{N}$ together with $\infty$ and $-\infty$, as well as the set of all positive real numbers $\mathbb{R}_+$ together with $\infty$ and $-\infty$ form semirings with respect to maximum and addition operation. These semirings are called *arctic semirings*.

*Example 1.8.* The real unit interval $[0, 1]$, with respect to the maximum operation and the ordinary multiplication, forms a semiring called the *Viterbi semiring*.

*Example 1.9.* The two element set $\{0, 1\}$ forms a semiring with respect to the maximum and minimum operations. This semiring is called the *Boolean semiring* and has 0 and 1 as its neutral elements.

*Example 1.10.* Given a non empty set $X$, a *semiring of formal languages* over $X$ is a semiring $(\mathscr{P}(X^*), \cup, \cdot, \emptyset, \{\varepsilon\})$, where $\mathscr{P}(X^*)$ is the set of all subsets of the free monoid $X^*$, $\cup$ is the classical union and $\cdot$ is the concatenation operation.

An algebra is *locally finite* if any of its finitely generated subalgebras is finite. It can be shown that a semiring $(S, +, \cdot, 0, 1)$ is locally finite if and only if

both monoids $(S,+,0)$ and $(S,\cdot,1)$ are locally finite. Indeed, if $(S,+,0)$ and $(S,\cdot,1)$ are locally finite and $U$ is a finite subset of $S$, then the submonoid $V$ of $(S,\cdot,1)$ generated by $U$ is finite and the submonoid $W$ of $(S,+,0)$ generated by $V$ is also finite. Now, it is easy to check that $W \cdot W \subseteq W$ and we deduce that the subsemiring of $(S,+,\cdot,0,1)$ generated by $U$ is the finite set $W$.

For instance, if both addition and multiplication are commutative and idempotent, then the semiring is locally finite. Consequently, any bounded distributive lattice $(L,\vee,\wedge,0,1)$ is a locally finite semiring. In particular, the chain $([0,1],\max,\min,0,1)$ and any Boolean algebra are locally finite. Further, the Łukasiewicz semiring $([0,1],\max,\otimes,0,1)$ is locally finite, since its additive and multiplicative monoids are commutative and locally finite.

In the further text, if not noted otherwise, $K$ will be a strong bimonoid.

An *$K$-valued vector* over a non-empty finite set $A$ is a mapping $\nu : A \to S$. Ordinary crisp subsets of $A$ can be viewed as $K$-valued vectors of $A$, where $K$ is the Boolean semiring $(\{0,1\},\max,\min,0,1)$.

An *$K$-valued matrix* over non-empty finite sets $A$ and $B$ (in this order) is any mapping $\mu : A \times B \to S$, i.e. , any $K$-valued vector of $A \times B$. In particular, an $K$-valued matrix over $A$ is any function from $A \times A$ to $K$, i.e., any $K$-valued vector over $A \times A$.

For non-empty finite sets $A$, $B$ and $C$, and $K$-valued matrices $\mu_1 : A \times B \to S$ and $\mu_2 : B \times C \to K$, their *product* $\mu_1 \cdot \mu_2$ is an $K$-valued matrix over $A$ and $C$ defined by

$$(\mu_1 \cdot \mu_2)(a,c) = \sum_{b \in B} \mu_1(a,b) \cdot \mu_2(b,c), \tag{1.1}$$

for all $a \in A$ and $c \in C$. Next, if $\nu_1 : A \to K$, $\mu : A \times B \to S$ and $\nu_2 : B \to K$, the products $\nu_1 \cdot \mu$ and $\mu \cdot \nu_2$ are $K$-valued vectors over $B$ and $A$, respectively, which are defined by

$$(\nu_1 \cdot \mu)(b) = \sum_{a \in A} \nu_1(a) \cdot \mu(a,b), \quad (\mu \cdot \nu_2)(a) = \sum_{b \in B} \mu(a,b) \cdot \nu_2(b), \tag{1.2}$$

for every $a \in A$ and $b \in B$.

The product of two $K$-valued vectors $\nu_1$ and $\nu_2$ of $A$ is an element of $K$ given by:

$$\nu_1 \cdot \nu_2 = \sum_{a \in A} \nu_1(a) \otimes \nu_2(a). \tag{1.3}$$

It is interesting to note that, in the case when $K$ is a semiring, due to distributivity, we have that all of these products are associative.

## 1.2. Ordered sets and lattices

A *binary relation* on a non-empty set $A$ is a subset $R$ of the Cartesian product $A \times A = \{(x,y) | x,y \in A\}$. We say that $x,y \in A$ are *R-related* whenever $(x,y) \in R$, this often being written in the equivalent form $x\,R\,y$. Generally, there are many properties that binary relations may satisfy on a given set $A$. In particular, a binary relation $R$ on $A$ is said to be

(1) *reflexive*, if it satisfies $(x,x) \in R$, for all $x \in A$;
(2) *symmetric*, if $(x,y) \in R$ implies $(y,x) \in R$, for all $x,y \in A$;
(3) *anti-symmetric*, if $(x,y) \in R$ and $(y,x) \in R$ implies $x = y$, for all $x,y \in A$;
(4) *transitive*, if $(x,y) \in R$ and $(y,z) \in R$ implies $(x,z) \in R$, for all $x,y,z \in A$.

The *dual* of a binary relation $R$ is the relation $R^{-1}$ given by

$$(x,y) \in R^{-1} \quad \Leftrightarrow \quad (y,x) \in R.$$

A binary relation which is reflexive, symmetric and transitive is called an *equivalence relation*, whereas a binary relation which is reflexive, anti-symmetric and transitive is called a *partial order*, or briefly an *order*. We usually denote an order by the symbol $\leqslant$. It is traditional to write the expression $(x,y) \in \leqslant$ in the equivalent form $x \leqslant y$ which we read as "$x$ is less than or equal to $y$".

By a *partially ordered set*, or simply *ordered set* $(A, \leqslant)$ we shall mean a set $A$ on which there is defined an order $\leqslant$.

According to Birkhoff [13], the defining properties of an order occur in a fragmentary way in the work of Leibniz (circa 1690). The present formulation emerged from the work of Peirce [91], Schröder [101], and Hausdorff [62].

Examples of some of the most commonly used ordered sets are given below:

*Example 1.11.* Given an arbitrary set $A$, the relation of equality "= " is an order on $A$.

*Example 1.12.* On the set $\mathscr{P}(A)$ of all subsets of a non-empty set $A$ the relation $\subseteq$ of set inclusion is an order.

*Example 1.13.* The set of even positive integers may be ordered in the usual way, or by divisibility.

*Example 1.14.* If $(A_1, \leqslant_1)$, . . . , $(A_n, \leqslant_n)$ are ordered sets then the Cartesian product set $A_1 \times \cdots \times A_n$ can be given the Cartesian order $\leqslant$ defined by

$$(x_1,...,x_n) \leqslant (y_1,...,y_n) \quad \Leftrightarrow \quad x_i \leqslant_i y_i,\ i \in \{1,...,n\}.$$

In the sequel, we shall denote the dual of an order $\leqslant$ on $A$ by the symbol $\geqslant$ which we read as "greater than or equal to". Then the ordered set $(A, \geqslant)$ is called the dual of $(A, \leqslant)$, hence to every statement that concerns an order on

a set $A$ there is a dual statement that concerns the corresponding dual order on $A$. This is the basis of the useful

*Principle of Duality*: To every theorem that concerns an ordered set $A$ there is a corresponding theorem that concerns the dual ordered set. This is obtained by replacing each statement that involves $\leqslant$, explicitly or implicitly, by its dual.

In what follows we shall use the notation $x < y$ to mean $x \leqslant y$ and $x \neq y$. Note that the relation $<$ thus defined is transitive but is not an order since it fails to be reflexive; moreover, $x < y$ and $y < x$ are incompatible. We denote the dual of the relation $<$ by the symbol $>$. An order $\leqslant$ on $A$ is a *linear order* on $A$ if for every $x, y \in A$ holds $x \leqslant y$ or $x \leqslant y$. In this case $A$ is a *linearly ordered set*.

If $(A, \leqslant)$ is an ordered set then by the *greatest element* of $A$ we mean an element $x \in A$ such that $y \leqslant x$ for every $y \in A$. The greatest element, when it exists, is unique. In fact, if $x, y$ are both greatest elements of $A$ then on the one hand $y \leqslant x$ and on the other $x \leqslant y$ whence, by the anti-symmetric property of $\leqslant$, we have $x = y$. The dual notion is that of the *least element*, namely an element $z \in A$ such that $z \leqslant y$ for every $y \in A$. By the above and the Principle of Duality, we can assert immediately that the least element, when it exists, is unique. The element $x \in A$ is called the *maximal element* of the set $A$, if $x \leqslant y$ implies $x = y$ for every $y \in A$, that is, if there is no element in the set $A$ which is strictly larger than $x$. The dual notion is that of the *minimal element* of the set $A$, i.e. the element $x \in A$ is said to be minimal element of $A$ if $y \leqslant x$ implies $x = y$ for every $y \in A$.

Next, let $B$ be a non-empty subset of an ordered set $A$. The *upper bound* of the set $B$ is an element $x \in A$ such that $y \leqslant x$, for every $y \in B$. The *lower bound* of the set $B$ is an element $x \in A$ such that $x \leqslant y$, for every $y \in B$. The *least upper bound* or the *supremum* of the set $B$ is the element $x \in A$, which is the least element in the set of all upper bounds of $B$, in other words, which is the upper bound of $B$ and for any upper bound $y$ of the set $B$ there holds $x \leqslant y$. Dually, the *greatest lower bound* or the *infimum* of the set $B$ is the element $x \in A$, which is the greatest element in the set of all lower bounds of $B$, that is, which is the lower bound of $B$ and for any lower bound $y$ of the set $B$ there holds $y \leqslant x$. The supremum of the set $B$, if it exists, is denoted by $\bigvee B$, whereas the infimum of $B$, if it exists, is usually denoted by $\bigwedge B$. If $B = \{x_i | i \in I\}$, instead of $\bigvee B$ and $\bigwedge B$ we can write, respectively:

$$\bigvee_{i \in I} x_i \quad \text{and} \quad \bigwedge_{i \in I} x_i.$$

A *lattice* is an ordered set such that every two-element subset has the supremum and the infimum. Given a lattice $L$, we can define two binary operations $\bigvee$ and $\bigwedge$ on $L$, by:

$$\bigvee(x, y) = x \vee y \quad \text{and} \quad \bigwedge(x, y) = x \wedge y.$$

These operations are called *union* and *intersection*, respectively.

**Theorem 1.1.** *Let $L$ be a lattice. Then for every $x, y, z \in L$ the following holds:*

*(L1)* $x \wedge x = x, \quad x \vee x = x$ *(idempotents );*
*(L2)* $x \wedge y = y \wedge x, \quad x \vee y = y \vee x$ *(commutativity);*
*(L3)* $(x \wedge y) \wedge z = x \wedge (y \wedge z), \quad (x \vee y) \vee z = x \vee (y \vee z)$ *(associativity);*
*(L4)* $x \wedge (x \vee y) = x, \quad x \vee (x \wedge y) = x$ *(absorption).*

The conditions $(L1) - (L4)$ of the Theorem 1.1 are called *the lattice axioms*.
A subset $A$ of a lattice $L$ is called a *sublattice* if $x \wedge y \in A$ and $x \vee y \in A$, for all elements $x, y \in A$.

For a lattice $L$ and an element $x \in L$, sublattices

$$[x) = \{y \in L \,|\, x \leqslant y\} \quad \text{i} \quad (x] = \{y \in L \,|\, y \leqslant x\}$$

are half-open intervals of the lattice $L$, and for all $x, y \in L$ sublattices

$$(x, y) = \{z \in L \,|\, x < z < y\} \quad \text{i} \quad [x, y] = \{z \in L \,|\, x \leqslant z \leqslant y\}$$

are an open interval and a closed interval, respectively.

An algebra with one binary idempotent, commutative and associative operation is called a *semilattice*. If a non empty set $L$ together with arbitrary elements $x, y \in L$ contain $x \wedge y$, respectively $x \vee y$ then $L$ is called $\wedge$-*semilattice* or *lower semilattice*, i.e. $\vee$-*semilattice* or *upper semilattice*.

A non-empty subset $I$ of a lattice $L$ is called an *ideal* if:

(1) for all $a, x \in L$, by $x \leqslant a$ and $a \in I$ it follows $x \in I$;
(2) $x \vee y \in I$, for all $x, y \in L$.

It can easily be shown that a subset $I$ is an ideal of the lattice $L$ if the following is true:

$$x \wedge y \in I \quad \text{if and only if both } x \text{ and } y \text{ are elements of } I.$$

The dual notion to the notion of an ideal is the *dual ideal*. Namely, a non-empty subset $D$ of a lattice $L$ is a *dual ideal* if:

(1) for all $a, x \in L$, by $a \leqslant x$ and $a \in D$ it follows $x \in D$;
(2) $x \wedge y \in I$, for all $x, y \in L$.

Let $x \in L$. Notice that the half-open intervals $(x]$ and $[x)$, are an ideal and a dual ideal of the lattice $L$, respectively, and they are called *the principal ideal generated by* $x$ and *the principal dual ideal generated by* $x$ .

The least element of the lattice $L$, if it exists, is denoted by $0$, and the greatest element, if it exists, is denoted by $1$. A *bounded lattice* is a lattice that has the greatest element $1$ and the least element $0$.

It can easily be shown that for every lattice $L$ the following conditions are equivalent:

(L5) $x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$, for all $x, y, z \in L$;
(L5′) $x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$, for all $x, y, z \in L$.

A lattice which satisfies any of the previous conditions is called a *distributive* lattice.

Let $L$ be a bounded lattice with 0 and 1. An element $y \in L$ is called a *complement* of an element $x \in L$ if

$$x \wedge y = 0, \quad x \vee y = 1.$$

In that case, the element $x \in L$ is a complement of the element $y \in L$. In other words, the relation "to be a complement" is a symmetric relation. In every distributive lattice it can easily be shown that every element $x \in L$ has at most one complement, which will be denoted with $x'$.

A bounded distributive lattice in which every element has a complement is called a *Boolean lattice*. The mapping $x \to x'$ is a unary operation on $L$ called the *complement operation*.

As we have maintained before, every finite subset of a lattice has a supremum and infimum. However, it does not have to be the case when the subset of a lattice is infinite. A lattice in which every subset (finite or infinite) has a supremum and an infimum is called a *complete lattice*. Clearly, every such a lattice is bounded. A subset $K$ of a complete lattice $L$ is a *complete sublattice* of $L$ if the supremum and the infimun (in $L$) of every non-empty subset of $K$ belong to $K$.

Next, we give some examples of ordered set which are lattices:

*Example 1.15.* A singleton set $A = \{a\}$ is a lattice under the only possible order on $A$. This is a trivial lattice. Any lattice with more than one element is a nontrivial lattice.

*Example 1.16.* The ordered set $(\mathscr{P}(A), \subseteq)$ is a bounded lattice, for every set $A$. The supremum and infimum operations are union and intersection, respectively.

*Example 1.17.* The set $N$ of natural numbers is a lattice under division, where

$$x \wedge y = gcd(x, y) \quad \text{and} \quad x \vee y = lcm(x, y).$$

## 1.3. Complete residuated lattices

Contrary to classical logic, which is bivalent, the fuzzy logic is the logic of graded truth. In classical logic, each proposition is either assigned truth degree 1 (true) or truth degree 0 (false), whereas, in fuzzy logic, each proposition is assigned a truth degree taken from some scale $L$ of truth degrees. Scale $L$ is partially ordered and contains 0 and 1 as its boundary elements.

That is, $L$ is equipped with a partial order $\leqslant$ and we have $0 \leqslant a \leqslant 1$ for each $a \in L$. Elements a from $L$ are called truth degrees. If propositions $\varphi$ and $\psi$ are assigned truth degrees $a$ and $b$, then $a \leqslant b$ means that we consider $\varphi$ less true than $\psi$. That is why it is needed for $L$ to be partially ordered. It is in this sense that fuzzy logic is a logic of graded truth.

The set $L$ needs to be equipped with logical connectives. That is, as in classical logic, we need binary functions $\otimes : L \times L \to L$ of conjunction, $\to : L \times L \to L$ of implication, etc. Just as classical logic, fuzzy logic is also truth-functional. This means that a truth degree of a composed proposition is computed from truth degrees of its constituent propositions using logical connectives. The only difference is that in fuzzy logic, logical connectives are not uniquely given. In fuzzy logic, it is useful to postulate desirable properties of connectives and consider as a "good connective" any one satisfying the properties. For instance, a desirable property of conjunction is monotony – the more true the propositions, the more true their conjunction. In the following, we proceed this way to justify a particular structure of truth degrees. The structure will be called a complete residuated lattice and will play the role of a basic structure of truth degrees in our investigation. Our justification of complete residuated lattices as suitable structures of truth degrees is due to Goguen [45].

As stated before, a set $L$ of truth degrees should be equipped with a partial order $\leqslant$ and bounded by 0 and 1. Furthermore, we want L to have arbitrary infimum and supremum, i.e. we want $(L, \leqslant, 0, 1)$ to be a complete lattice. This is due to the fact that we need supremum (infimum) for modeling existential (universal) quantifier. Basic requirements regarding conjunction $\otimes$ are that $(L, \otimes, 1)$ is a commutative monoid. This means that $\otimes$ is a binary operation on $L$ which is commutative, associative, and 1 is a neutral element for $\otimes$. We proceed by (truth function of) implication which we denote by $\to$. Several desirable properties of implication follow from a simple condition called adjointness. Adjointness itself says that for each $a, b, c \in L$ we have $a \otimes b \leqslant c$ if and only if $a \leqslant b \to c$. This condition follows from how modus ponens should behave in fuzzy setting. Recall first that in classical logic, modus ponens is an inference rule saying: if $\varphi$ is valid and $\varphi$ implies $\psi$ is valid then we may infer that $\psi$ is valid. An appropriate formulation of modus ponens in fuzzy setting is the following: if $\varphi$ is valid in degree at least $a$ and $\varphi \to \psi$ is valid in degree at least $b$ then we may infer that $\psi$ is valid in degree at least $a \times b$.

A structure consisting of a set $L$ of truth degrees, a partial order $\leqslant$ bounded by 0 and 1, and functions $\otimes$ and $\to$ which satisfy the above conditions is called a complete residuated lattice and plays the role of a basic structure of truth degrees in our investigation.

A *residuated lattice* is an algebra $\mathscr{L} = (L, \wedge, \vee, \otimes, \to, 0, 1)$ such that:

**(L1)** $(L, \wedge, \vee, 0, 1)$ is a lattice with the least element 0 and the greatest element 1,

**(L2)** $(L, \otimes, 1)$ is a commutative monoid with the unit 1,

**(L3)** $\otimes$ and $\rightarrow$ form an adjoint pair: for each $a,b,c \in L$ we have $a \otimes b \leqslant c$ if and only if $a \leqslant b \rightarrow c$.

If, in addition, $(L,\wedge,\vee,0,1)$ is a complete lattice, then $\mathscr{L}$ is called a *complete residuated lattice*.

The operation $\otimes$ is called *multiplication*, while the operation $\rightarrow$ is called *residuum*. These operations are intended for modeling the conjunction and implication of the corresponding logical calculus.

An operation $\leftrightarrow$ called *biresiduum* is given by:

$$x \leftrightarrow y = (x \rightarrow y) \wedge (y \rightarrow x). \tag{1.4}$$

It is used for modeling the equivalence of truth values.

The operation *negation*, denoted by $\neg$, is defined by:

$$\neg x = x \rightarrow 0, \tag{1.5}$$

and is used for modeling the complement of a truth value.

Most important examples of complete residuated lattices are those with the universe $L$ being a real unit interval $[0,1]$ where:

$$x \wedge y = \min(x,y) \quad \text{and} \quad x \vee y = \max(x,y)$$

and multiplication and residuum operations are defined as follows:
the *Goguen* (*product*) *structure*:

$$x \otimes y = x \cdot y, \qquad x \rightarrow y = \begin{cases} 1, & \text{if } x \leqslant y, \\ y/x, & \text{otherwise,} \end{cases}$$

the *Gödel structure*:

$$x \otimes y = \min(x,y), \qquad x \rightarrow y = \begin{cases} 1, & \text{if } x \leqslant y, \\ y, & \text{otherwise,} \end{cases}$$

and the *Łukasiewicz structure*:

$$x \otimes y = \max(x+y-1,0), \quad x \rightarrow y = \min(1-x+y,1).$$

Another important set of truth values is the set $\{a_0,a_1,\ldots,a_n\}$, with

$$a_k \otimes a_l = a_{\max(k+l-n,0)} \quad \text{and} \quad a_k \rightarrow a_l = a_{\min(n-k+l,n)},$$

where $0 = a_0 < \cdots < a_n = 1$.

A special case of the latter algebras is the two-element Boolean algebra of classical logic with the support $\{0,1\}$. The only adjoint pair on the two-element Boolean algebra consists of the classical conjunction and implication operations. This structure of truth values we call the *Boolean structure*.

All of the structures mentioned before, the Łukasiewicz, Goguen and Gödel structures are residuated lattices induced by t-norms, a binary operation on $[0,1]$ which is associative, commutative, monotone and where 1 is unit element. In the case of the Łukasiewicz, Goguen and Gödel structures $\otimes$ is a t-norm. Generally, an algebra $([0,1],\vee,\wedge,\otimes,\rightarrow,0,1)$ is a complete residuated lattice if and only if $\otimes$ is a left-continuous t-norm, i.e. a t-norm satisfying

$$\lim_{n\to\infty}(a_n\otimes b)=(\lim_{n\to\infty}a_n)\otimes b,$$

and then the residuum is defined by

$$x\rightarrow y=\bigvee\{u\in[0,1]\,|\,u\otimes x\leqslant y\}.$$

Here we recall some important truth structures, which are residuated lattices, but satisfy some additional conditions.

A residuated lattice $\mathscr{L}$ satisfying $x\otimes y=x\wedge y$ is called a *Heyting algebra*. Moreover, if $\mathscr{L}$ is a complete lattice, then it is called a *complete Heyting algebra*, and if the partial order in $\leqslant$ in $\mathscr{L}$ is linear, then $\mathscr{L}$ is a *linearly ordered Heyting algebra*. The most important example of a linearly ordered Heyting algebra is the real unit interval $[0,1]$ with the Gödel pair of adjoint operations, i.e. with the standard minimum t-norm.

A *BL-algebra* (*basic logic algebra*) is a residuated lattice satisfying the condition $a\wedge b=a\otimes(a\rightarrow b)$ (*divisibility*) and $(a\rightarrow b)\vee(b\rightarrow a)=1$ (*prelinearity*). An *MV-algebra* (*multi-valued algebra*) is a BL-algebra in which $a=\neg\neg a$ holds (the *double negation* is allowed). A *P-algebra* (*product algebra*) is a BL-algebra which satisfies $(c\rightarrow 0)\rightarrow 0\leqslant((a\otimes c)\rightarrow(b\otimes c))\rightarrow(a\rightarrow b)$ and $a\wedge(a\rightarrow 0)=0$. A *G-algebra* (*Gödel algebra*) is a BL-algebra which satisfies $a\otimes a=a$ (*idempotency*). A *Boolean algebra* is a residuated lattice which is both a Heyting algebra and an MV-algebra.

Note that every Gödel algebra, and hence, the Gödel structure, is locally finite, whereas the product structure is not locally finite.

In what follows, we recall some basic properties of residuated lattices:

**Theorem 1.2.** *For every residuated lattice the following holds:*

$$x\otimes 0=0\otimes x=0, \tag{1.6}$$
$$x\otimes y\leqslant x,\quad x\leqslant y\rightarrow x, \tag{1.7}$$
$$x\otimes y\leqslant x\wedge y, \tag{1.8}$$
$$x\rightarrow x=1,\quad x\rightarrow 1=1,\quad 1\rightarrow x=x, \tag{1.9}$$
$$0\rightarrow x=1, \tag{1.10}$$
$$y\leqslant x\rightarrow(x\otimes y),\qquad x\leqslant(x\rightarrow y)\rightarrow y, \tag{1.11}$$
$$x\otimes(x\rightarrow y)\leqslant y, \tag{1.12}$$
$$(x\rightarrow y)\otimes(y\rightarrow z)\leqslant(x\rightarrow z), \tag{1.13}$$

$$x \leqslant y \quad \Leftrightarrow \quad x \to y = 1, \tag{1.14}$$
$$x \to y \text{ is the greatest element of } \{z \mid x \otimes z \leqslant y\} \tag{1.15}$$
$$x \otimes y \text{ is the least element of } \{z \mid x \leqslant y \to z\}. \tag{1.16}$$

**Theorem 1.3.** *In every residuated lattice the following inequalities hold:*

$$x \to y \leqslant (x \wedge z) \to (y \wedge z), \tag{1.17}$$
$$x \to y \leqslant (x \vee z) \to (y \vee z), \tag{1.18}$$
$$x \to y \leqslant (x \otimes z) \to (y \otimes z), \tag{1.19}$$
$$x \to y \leqslant (y \to z) \to (x \to z), \tag{1.20}$$
$$x \to y \leqslant (z \to x) \to (z \to y). \tag{1.21}$$

**Theorem 1.4.** *The following assertions hold in every residuated lattice:*

$$x \to y = ((x \to y) \to y) \to y, \tag{1.22}$$
$$x \otimes (x \to y) = y \quad \Leftrightarrow \quad (\exists z)(x \otimes z = y), \tag{1.23}$$
$$x \to (x \otimes y) = y \quad \Leftrightarrow \quad (\exists z)(x \to z = y), \tag{1.24}$$
$$(y \to x) \to x = y \quad \Leftrightarrow \quad (\exists z)(z \to x = y), \tag{1.25}$$
$$(x \wedge y) \otimes (x \vee y) \leqslant x \otimes y, \tag{1.26}$$
$$x \vee y \leqslant ((x \to y) \to y) \wedge ((y \to x) \to x), \tag{1.27}$$
$$x \wedge y \geqslant x \otimes (x \to y), \tag{1.28}$$
$$x \otimes (y \to z) \leqslant y \to (x \otimes z). \tag{1.29}$$

The next theorem shows that with respect to $\leqslant$, the operation $\otimes$ is isotonic in both arguments, the operation $\to$ is isotonic in the second and antitonic in the first argument.

**Theorem 1.5.** *In every residuated lattice the following holds:*

$$y_1 \leqslant y_2 \Rightarrow x \otimes y_1 \leqslant x \otimes y_2, \tag{1.30}$$
$$y_1 \leqslant y_2 \Rightarrow x \to y_1 \leqslant x \to y_2, \tag{1.31}$$
$$x_1 \leqslant x_2 \Rightarrow x_2 \to y \leqslant x_1 \to y. \tag{1.32}$$

The next theorem describes a relationship between operations $\vee$ and $\wedge$, for any family of residuated lattice elements, and operations $\otimes$ and $\to$.

**Theorem 1.6.** *The following assertions hold for every index set I:*

$$x \otimes \Big(\bigvee_{i \in I} y_i\Big) = \bigvee_{i \in I} (x \otimes y_i), \tag{1.33}$$
$$x \to \Big(\bigwedge_{i \in I} y_i\Big) = \bigwedge_{i \in I} (x \to y_i), \tag{1.34}$$

$$\Big(\bigvee_{i\in I} x_i\Big) \to y = \bigwedge_{i\in I} (x_i \to y), \tag{1.35}$$

$$\bigvee_{i\in I} (x_i \to y) = \Big(\bigwedge_{i\in I} x_i\Big) \to y, \tag{1.36}$$

$$x \otimes \bigwedge_{i\in I} y_i \leqslant \bigwedge_{i\in I} (x \otimes y_i). \tag{1.37}$$

$$\bigwedge_{i\in I} (x_i \to y_i) \leqslant \Big(\bigwedge_{i\in I} x_i\Big) \to \Big(\bigwedge_{i\in I} y_i\Big) \tag{1.38}$$

$$\bigwedge_{i\in I} (x_i \to y_i) \leqslant \Big(\bigvee_{i\in I} x_i\Big) \to \Big(\bigvee_{i\in I} y_i\Big) \tag{1.39}$$

$$\bigvee_{i\in I} (x \to y_i) \leqslant x \to \Big(\bigwedge_{i\in I} y_i\Big) \tag{1.40}$$

## 1.4. Fuzzy sets and fuzzy relations

In the further text the complete residuated lattice $\mathscr{L}$ will be a structure of truth values.

A *fuzzy subset* of a set $A$ *over* $\mathscr{L}$, or simply a *fuzzy subset* of $A$, is any mapping from $A$ into $L$. Ordinary crisp subsets of $A$ are considered as fuzzy subsets of $A$ taking membership values in the set $\{0,1\} \subseteq L$. Let $f$ and $g$ be two fuzzy subsets of $A$. The *equality* of $f$ and $g$ is defined as the usual equality of mappings, i.e., $f = g$ if and only if $f(x) = g(x)$, for every $x \in A$. The *inclusion* $f \leqslant g$ is also defined pointwise: $f \leqslant g$ if and only if $f(x) \leqslant g(x)$, for every $x \in A$. Endowed with this partial order the set $\mathscr{F}(A)$ of all fuzzy subsets of $A$ forms a complete residuated lattice, in which the meet (intersection) $\bigwedge_{i\in I} f_i$ and the join (union) $\bigvee_{i\in I} f_i$ of an arbitrary family $\{f_i\}_{i\in I}$ of fuzzy subsets of $A$ are mappings from $A$ to $L$ defined by

$$\left(\bigwedge_{i\in I} f_i\right)(x) = \bigwedge_{i\in I} f_i(x), \qquad \left(\bigvee_{i\in I} f_i\right)(x) = \bigvee_{i\in I} f_i(x),$$

and the *product* $f \otimes g$ is a fuzzy subset defined by $f \otimes g(x) = f(x) \otimes g(x)$, for every $x \in A$. The *crisp part* of a fuzzy subset $f \in \mathscr{F}(A)$ is a crisp subset $\widehat{f} = \{a \in A \mid f(a) = 1\}$ of $A$.

A *fuzzy relation* between sets $A$ and $B$ (in this order) is any mapping from $A \times B$ to $L$, i.e. , any fuzzy subset of $A \times B$, and the equality, inclusion (ordering), joins and meets of fuzzy relations are defined as for fuzzy sets. The set of all fuzzy relations between $A$ and $B$ will be denoted by $\mathscr{R}(A,B)$. In particular, a fuzzy relation on a set $A$ is any function from $A \times A$ to $L$, i.e., any fuzzy subset of $A \times A$. The set of all fuzzy relations on $A$ will be denoted by $\mathscr{R}(A)$. The *inverse*(*converse*, or *transpose*) of a fuzzy relation $\varphi \in \mathscr{R}(A,B)$ is

a fuzzy relation $\varphi^{-1} \in \mathscr{R}(B,A)$ defined by $\varphi^{-1}(b,a) = \varphi(a,b)$, for all $a \in A$ and $b \in B$. A crisp relation is a fuzzy relation which takes values only in the set $\{0,1\}$, and if $\varphi$ is a crisp relation of $A$ to $B$, then expressions "$\varphi(a,b) = 1$" and "$(a,b) \in \varphi$" will have the same meaning.

For arbitrary non-empty sets $A$, $B$ and $C$, and fuzzy relations $\varphi \in \mathscr{R}(A,B)$ and $\psi \in \mathscr{R}(B,C)$, their *composition* is a fuzzy relation $\varphi \circ \psi \in \mathscr{R}(A,C)$ given by

$$(\varphi \circ \psi)(a,c) = \bigvee_{b \in B} \varphi(a,b) \otimes \psi(b,c), \tag{1.41}$$

for all $a \in A$ and $c \in C$. Further, for $f \in \mathscr{F}(A)$, $\varphi \in \mathscr{R}(A,B)$ and $g \in \mathscr{F}(B)$, the compositions $f \circ \varphi$ and $\varphi \circ g$ are fuzzy subsets of $B$ and $A$, respectively, which are defined by

$$(f \circ \varphi)(b) = \bigvee_{a \in A} f(a) \otimes \varphi(a,b), \quad (\varphi \circ g)(a) = \bigvee_{b \in B} \varphi(a,b) \otimes g(b), \tag{1.42}$$

for every $a \in A$ and $b \in B$.

In particular, for fuzzy subsets $f$ and $g$ of $A$ we write

$$f \circ g = \bigvee_{a \in A} f(a) \otimes g(a). \tag{1.43}$$

Let $A$, $B$, $C$ and $D$ be non-empty sets. Then for any $\varphi_1 \in \mathscr{R}(A,B)$, $\varphi_2 \in \mathscr{R}(B,C)$ and $\varphi_3 \in \mathscr{R}(C,D)$ we have:

$$(\varphi_1 \circ \varphi_2) \circ \varphi_3 = \varphi_1 \circ (\varphi_2 \circ \varphi_3), \tag{1.44}$$

and for $\varphi_0 \in \mathscr{R}(A,B)$, $\varphi_1, \varphi_2 \in \mathscr{R}(B,C)$ and $\varphi_3 \in \mathscr{R}(C,D)$ we have

$$\varphi_1 \leqslant \varphi_2 \quad \text{implies} \quad \varphi_0 \circ \varphi_1 \leqslant \varphi_0 \circ \varphi_2 \quad \text{and} \quad \varphi_1 \circ \varphi_3 \leqslant \varphi_2 \circ \varphi_3. \tag{1.45}$$

Further, for any $\varphi \in \mathscr{R}(A,B)$, $\psi \in \mathscr{R}(B,C)$, $f \in \mathscr{F}(A)$, $g \in \mathscr{F}(B)$ and $h \in \mathscr{F}(C)$ we can easily verify that

$$(f \circ \varphi) \circ \psi = f \circ (\varphi \circ \psi), \quad (f \circ \varphi) \circ g = f \circ (\varphi \circ g), \quad (\varphi \circ \psi) \circ h = \varphi \circ (\psi \circ h), \tag{1.46}$$

and consequently, the parentheses in (1.46) can be omitted, as well as the parentheses in (1.44). Finally, for all $\varphi, \varphi_i \in \mathscr{R}(A,B)$, $(i \in I)$ and $\psi, \psi_i \in \mathscr{R}(B,C)$, $(i \in I)$ we have that

$$(\varphi \circ \psi)^{-1} = \varphi^{-1} \circ \psi^{-1} \tag{1.47}$$

$$\varphi \circ \Big(\bigvee_{i \in I} \psi_i\Big) = \bigvee_{i \in I} (\varphi \circ \psi_i), \quad \Big(\bigvee_{i \in I} \varphi_i\Big) \circ \psi = \bigvee_{i \in I} (\varphi_i \circ \psi) \tag{1.48}$$

$$\Big(\bigvee_{i \in I} \varphi_i\Big)^{-1} = \bigvee_{i \in I} \varphi_i^{-1}. \tag{1.49}$$

We note that if $A$, $B$ and $C$ are finite sets of cardinality $|A| = k$, $|B| = m$ and $|C| = n$, then $\varphi \in \mathscr{R}(A,B)$ and $\psi \in \mathscr{R}(B,C)$ can be treated as $k \times m$ and $m \times n$ fuzzy matrices over $\mathscr{L}$, and $\varphi \circ \psi$ is the matrix product. Analogously, for $f \in \mathscr{F}(A)$ and $g \in \mathscr{F}(B)$ we can treat $f \circ \varphi$ as the product of a $1 \times k$ matrix $f$ and a $k \times m$ matrix $\varphi$ (vector-matrix product), $\varphi \circ g$ as the product of an $k \times m$ matrix $\varphi$ and an $m \times 1$ matrix $g^t$, the transpose of $g$ (matrix-vector product), and $f \circ g$ as the scalar product of vectors $f$ and $g$.

Let $\varphi, \psi \in \mathscr{R}(A)$. The *right residual* of $\psi$ by $\varphi$ is a fuzzy relation $\varphi \backslash \psi \in \mathscr{R}(A)$ defined by

$$(\varphi \backslash \psi)(a,b) = \bigwedge_{c \in A} \varphi(c,a) \to \psi(c,b),$$

and the *left residual* of $\psi$ by $\varphi$ is a fuzzy relation $\psi / \varphi \in \mathscr{R}(A)$ defined by

$$(\psi / \varphi)(a,b) = \bigwedge_{c \in C} \varphi(b,c) \to \psi(a,c),$$

for all $a, b \in A$. Further, for $f, g \in \mathscr{F}(A)$, the *right residual* of $g$ by $f$ is a fuzzy relation $f \backslash g \in \mathscr{F}(A)$ and the *left residual* of $g$ by $f$ is a fuzzy relation $g / f \in \mathscr{F}(A)$ defined by

$$(f \backslash g)(a,b) = f(a) \to g(b), \qquad (g/f)(b,a) = f(a) \to g(b),$$

for all $a, b \in A$.

A fuzzy relation $R$ on a set $A$ is said to be:

(R) *reflexive* (or *fuzzy reflexive*) if $R(a,a) = 1$, for every $a \in A$;
(S) *symmetric* (or *fuzzy symmetric*) if $R(a,b) = R(b,a)$, for all $a, b \in A$;
(T) *transitive* (or *fuzzy transitive*) if $R(a,b) \otimes R(b,c) \leqslant R(a,c)$, for all $a, b, c \in A$.

It can easily be shown, that $R \circ R = R$ holds for any reflexive and transitive relation $R$ on $A$.

A reflexive, symmetric and transitive fuzzy relation on $A$ is called a *fuzzy equivalence*. With the respect to the inclusion of fuzzy relations, the set $\mathscr{E}(A)$ of all fuzzy equivalences on $A$ is a complete lattice, in which the meet coincide with the ordinary intersection of fuzzy relations, but in the general case, the join in $\mathscr{E}(A)$ does not coincide with the ordinary union of fuzzy relations.

A fuzzy equivalence $E$ on a set $A$ is called a *fuzzy equality* if $E(a,b) = 1$ implies $a = b$, for all $a, b \in A$. In other words, $E$ is a fuzzy equality if and only if its crisp part $\widehat{E}$ is a crisp equality.

The *equivalence class* of fuzzy relation $E$ on $A$ determined by $a \in A$ is the fuzzy subset $E_a$ of $A$ defined by

$$E_a(b) = E(a,b), \quad \text{for every } b \in A.$$

The set $A/E = \{E_a \mid a \in A\}$ is called the *factor set* of $A$ w.r.t. $E$ (cf. [11]). The *natural function* from $A$ to $A/E$ is the fuzzy relation $\varphi_E \in \mathscr{R}(A, A/E)$ defined by $\varphi_E(a, E_b) = E(a,b)$, for all $a, b \in A$.

For a fuzzy relation $R$ on a set $A$, a fuzzy relation $R^\infty$ on $A$ defined by

$$R^\infty = \bigvee_{n\in\mathbb{N}} R^n.$$

is the least transitive fuzzy relation on $A$ containing $R$, and it is called the *transitive closure* of $R$.

A fuzzy relation on a set $A$ which is reflexive and transitive is called a *fuzzy quasi-order*, and a reflexive and transitive crisp relation on $A$ is called a *quasi-order*. Like the set $\mathscr{E}(A)$, the set $\mathscr{Q}(A)$ of all fuzzy quasi-orders on $\mathscr{A}$ is also a complete lattice, in which the meet coincides with the ordinary intersection of fuzzy relations and, in the general case, the join in $\mathscr{Q}(A)$ does not coincide with the ordinary union of fuzzy relations. Namely, if $R$ is the join in $\mathscr{Q}(A)$ of a family $\{R_i\}_{i\in I}$ of fuzzy quasi-orders on $A$, then $R$ can be represented by:

$$R = \Big(\bigvee_{i\in I} R\Big)^\infty = \bigvee_{n\in\mathbb{N}} \Big(\bigvee_{i\in I} R\Big)^n.$$

For a fuzzy relation $R \in \mathscr{R}(A)$ and an element $a \in A$, the *R-afterset* of $a$ is the fuzzy set $R_a \in \mathscr{F}(A)$ defined by:

$$R_a(b) = R(a,b), \text{ for any } b \in A,$$

while the *R-foreset* of $a$ is the fuzzy set $R^a \in \mathscr{F}(A)$ defined by:

$$R^a(b) = R(b,a), \text{ for any } b \in A.$$

The set of all $R$-aftersets will be denoted by $A/R$, and the set of all $R$-foresets will be denoted by $A\backslash R$. Clearly, if $R$ is a fuzzy equivalence, then $A/R = A\backslash R$ is the set of all equivalence classes of $R$.

For a fuzzy quasi-order $R$ on $A$, a fuzzy relation $E_R$ defined by $E_R = R \wedge R^{-1}$ is a fuzzy equivalence on $A$, which is called the *natural fuzzy equivalence* of $R$. A fuzzy quasi-order $R$ on a set $A$ is a *fuzzy order* if $R(a,b) = R(b,a) = 1$ implies $a = b$, for all $a,b \in A$, i.e., if the natural fuzzy equivalence $E_R$ of $R$ is a fuzzy equality. Clearly, a fuzzy quasi-order $R$ is a fuzzy order if and only if its crisp part $\hat{R}$ is a crisp order.

The following theorem recalls some important features of quasi orders and natural equivalences.

**Theorem 1.7.** *Let $R$ be a fuzzy quasi-order on a set $A$ and $E$ the natural fuzzy equivalence of $R$. Then*

*(a) For arbitrary $a,b \in A$ the following conditions are equivalent:*

*(i)* $E(a,b) = 1$;
*(ii)* $E_a = E_b$;
*(iii)* $R_a = R_b$;
*(iv)* $R^a = R^b$.

*(b) Functions $R_a \to E_a$ of $A/R$ to $A/E$, and $R_a \to R^a$ of $A/R$ to $A \setminus R$, are bijective functions.*

For more information about fuzzy equivalence and fuzzy quasi-order relations we refer to [116].

# Chapter 2
# Weighted and fuzzy automata

Fuzzy and weighted automata are classical nondeterministic automata in which transitions, initial and final states take values from certain structures. These values may be used for modeling the probability or possibility of the successful transition, the cost paid when entering a certain state, etc. For weighted automata these values are called weights, and are usually taken from semirings, and for fuzzy automata they are called truth values, and are taken from certain ordered structures, the most often from lattice-ordered structures. It is worth noting that fuzzy and weighted automata have found application in many areas of computer science such as naturtal language processing, lexical analysis, description of natural and programming languages, learning systems, control systems, neural networks, clinical monitoring, pattern recognition, databases, discrete event systems, and many other areas.

The chapter is organized in the following way. In Section 2.1., we recall the notions of a formal power series and a weighted finite automaton over a strong bimonid. Furthermore, we consider tree different kinds of behaviors of a weighted automaton over a strong bimonoid. Section 2.2. is dedicated to fuzzy automata and fuzzy languages over a complete residuated lattice. In section 2.3. we deal with crisp-deterministic fuzzy and weighted automata together with language, respectively behavior of these automata. Finally, in Section 2.4., we consider afterset, foreset and factor fuzzy automaton.

## 2.1. Weighted finite automata and series over strong bimonoids

*In the rest of this section, if not noted otherwise, let $K$ be a strong bimonoid and $X$ an alphabet.*

A *formal power series* over $X$ and $K$, for short a *series*, is any mapping $\varphi : X^* \to K$. Instead of $\varphi(u)$ we write $(\varphi, u)$ for every $u \in X^*$. The set of all series over $X$ and $K$ is denoted by $K\langle\langle X^*\rangle\rangle$. The image of $\varphi$ is the set $\mathrm{Im}(\varphi) =$

$\{(\varphi,u) \in K \mid u \in X^*\}$, and the kernel of $\varphi$ is the relation $\ker\varphi$ on $X^*$ defined as follows: for $u,v \in X^*$ we set $(u,v) \in \ker\varphi$ if and only if $(\varphi,u) = (\varphi,v)$.

A *weighted finite automaton* (for short: *wfa*) over $X$ and $K$ is a quadruple $\mathscr{A} = (A,\delta,\sigma,\tau)$, where $A$ is a finite non-empty *set of states*, $\delta : A \times X \times A \to K$ is a *weighted transition function*, $\sigma : A \to K$ is an *initial weight vector* and $\tau : A \to K$ is a *final weight vector*. For each $x \in X$ we define a *weighted transition matrix* (or a *weighted transition relation*) $\delta_x : A \times A \to K$ by $\delta_x(a,b) = \delta(a,x,b)$, for all $a,b \in A$. If $\mathscr{A}$ is a wfa, we will write $|\mathscr{A}| = |A|$, i.e., $|\mathscr{A}|$ denotes the number of states of $\mathscr{A}$.

Due to lack of distributivity in $K$, we define the behavior of wfa in several different ways. In particular, we distinguish the *initial algebra semantics*, the *run semantics*, and the *transition semantics* of a weighted finite automaton [37].

*Initial algebra semantics*: For each $u \in X^*$ we define a vector $\sigma_u : A \to K$ inductively, as follows: $\sigma_\varepsilon = \sigma$, and for all $u \in X^*$ and $x \in X$ we set $\sigma_{ux} = \sigma_u \cdot \delta_x$. Clearly, if $u = x_1x_2\cdots x_n$, where $x_1,\ldots,x_n \in X$, then

$$\sigma_u = (\ldots((\sigma \cdot \delta_{x_1}) \cdot \delta_{x_2}) \cdot \ldots) \cdot \delta_{x_n}. \tag{2.1}$$

The *i-behavior* of $\mathscr{A}$ is the series $[\![\mathscr{A}]\!]_i$ in $K\langle\langle X^*\rangle\rangle$ defined by

$$([\![\mathscr{A}]\!]_i, u) = \sigma_u \cdot \tau = ((\ldots((\sigma \cdot \delta_{x_1}) \cdot \delta_{x_2}) \cdot \ldots) \cdot \delta_{x_n}) \cdot \tau\,. \tag{2.2}$$

for every $u = x_1x_2\cdots x_n \in X^*$, where $x_1,x_2,\ldots,x_n \in X$.

*Transition semantics*: For each $u \in X^*$ we define a matrix $\delta_u : A \times A \to K$ inductively, as follows: $\delta_\varepsilon$ is the unit matrix, and for all $u \in X^*$ and $x \in X$ we set $\delta_{ux} = \delta_u \cdot \delta_x$. In other words, if $u = x_1x_2\cdots x_n \in X^*$, where $x_1,x_2,\ldots,x_n \in X$, then

$$\delta_u = (\ldots((\delta_{x_1} \cdot \delta_{x_2}) \cdot \delta_{x_3}) \cdot \ldots) \cdot \delta_{x_n}. \tag{2.3}$$

Matrices $\delta_u$, $u \in X^*$, will be also called *weighted transition matrices*.

The *t-behavior* of $\mathscr{A}$ is the series $[\![\mathscr{A}]\!]_t$ in $K\langle\langle X^*\rangle\rangle$ defined by

$$([\![\mathscr{A}]\!]_t, u) = (\sigma \cdot \delta_u) \cdot \tau\,. \tag{2.4}$$

for every $u \in X^*$. If $K$ is a semiring, then the matrix product and matrix-vector product (whenever defined) are associative and the above parentheses can be deleted. In this case, the *t*-behavior becomes the *f-behavior* defined in [37].

*Run semantics*: The *r-behavior* of $\mathscr{A}$ is the series $[\![\mathscr{A}]\!]_r$ in $K\langle\langle X^*\rangle\rangle$ defined by

$$([\![\mathscr{A}]\!]_r, u) = \sum_{(a_0,a_1,\ldots,a_n)\in A^{n+1}} \sigma(a_0) \cdot \delta_{x_1}(a_0,a_1) \cdot \delta_{x_2}(a_1,a_2) \cdot \ldots \cdot \delta_{x_n}(a_{n-1},a_n) \cdot \tau(a_n)\,. \tag{2.5}$$

for every $u = x_1x_2\cdots x_n \in X^*$, where $x_1,x_2,\ldots,x_n \in X$. Here, a sequence $(a_0,a_1,\ldots,a_n) \in A^{n+1}$ is often called a *run* of $\mathscr{A}$ on $u = x_1x_2\cdots x_n$.

Let $s \in \{i,t,r\}$. A series $\varphi \in K\langle\langle X^* \rangle\rangle$ is called *s-recognizable* if there exists a wfa $\mathscr{A}$ over $X$ and $K$ such that $[\![\mathscr{A}]\!]_s = \varphi$. We say that two wfa $\mathscr{A}$ and $\mathscr{A}'$ over $X$ and $K$ are *s-equivalent* if $[\![\mathscr{A}]\!]_s = [\![\mathscr{A}']\!]_s$.

Given a series $\varphi \in K\langle\langle X^* \rangle\rangle$ and $v \in X^*$, we define the series $v^{-1}\varphi \in K\langle\langle X^* \rangle\rangle$ and $\varphi v^{-1} \in K\langle\langle X^* \rangle\rangle$ by letting $(v^{-1}\varphi, u) = (\varphi, vu)$ and $(\varphi v^{-1}, u) = (\varphi, uv)$ for $u \in X^*$. The series $v^{-1}\varphi$ is called a *left derivative*, and the series $\varphi v^{-1}$ a *right derivative* of $\varphi$ with respect to $v$. Derivatives of series have been well investigated for series over semirings (cf.[15]).

## 2.2. Fuzzy automata

In the further text, $\mathscr{L}$ will be a complete residuated lattice and $X$ will be an (finite) alphabet. A *fuzzy automaton over* $\mathscr{L}$ and $X$, or simply a *fuzzy automaton*, is a quadruple $\mathscr{A} = (A, \delta^A, \sigma^A, \tau^A)$, where $A$ is a non-empty set, called the *set of states*, $\delta^A : A \times X \times A \to L$ is a fuzzy subset of $A \times X \times A$, called the *fuzzy transition function*, and $\sigma^A : A \to L$ and $\tau^A : A \to L$ are the fuzzy subsets of $A$, called the *fuzzy set of initial states* and the *fuzzy set terminal states*, respectively. We can interpret $\delta^A(a,x,b)$ as the degree to which an input letter $x \in X$ causes a transition from a state $a \in A$ into a state $b \in A$, whereas we can interpret $\sigma^A(a)$ and $\tau^A(a)$ as the degrees to which $a$ is respectively an input state and a terminal state. For methodological reasons we sometimes allow the set of states $A$ to be infinite. A fuzzy automaton whose set of states is finite is called a *fuzzy finite automaton*.

In cases where it is clear which set of states is underlying for $\delta^A$, $\sigma^A$ and $\tau^A$, the superscript $A$ will be omitted, i.e., instead of $\mathscr{A} = (A, \delta^A, \sigma^A, \tau^A)$ we will write simply $\mathscr{A} = (A, \delta, \sigma, \tau)$.

Let $\mathscr{A} = (A, \delta, \sigma, \tau)$ be a fuzzy automaton over $X$ and $\mathscr{L}$, let $X^*$ denote the free monoid over $X$, and let $\varepsilon \in X^*$ be the empty word. The function $\delta$ can be extended up to a function $\delta^* : A \times X^* \times A \to L$ as follows: If $a, b \in A$, then

$$\delta^*(a,e,b) = \begin{cases} 1, & \text{if } a = b, \\ 0, & \text{otherwise,} \end{cases} \tag{2.6}$$

and if $a, b \in A$, $u \in X^*$ and $x \in X$, then

$$\delta^*(a,ux,b) = \bigvee_{c \in A} \delta^*(a,u,c) \otimes \delta(c,x,b). \tag{2.7}$$

By (1.33) and Theorem 3.1 [72] (see also [96, 100]), we have that

$$\delta^*(a,uv,b) = \bigvee_{c \in A} \delta^*(a,u,c) \otimes \delta^*(c,v,b), \tag{2.8}$$

for all $a,b \in A$ and $u,v \in X^*$.

If for any $u \in X^*$ we define a fuzzy relation $\delta_u$ on $A$ by

$$\delta_u(a,b) = \delta^*(a,u,b), \tag{2.9}$$

for all $a,b \in A$, called the *fuzzy transition relation* determined by $u$, then (2.8) can be written as

$$\delta_{uv} = \delta_u \circ \delta_v, \tag{2.10}$$

for all $u,v \in X^*$.

A *fuzzy language* in $X^*$ over $\mathscr{L}$, or briefly a *fuzzy language*, is any fuzzy subset of $X^*$, i.e., any function from $X^*$ into $L$. A *fuzzy language recognized by a fuzzy automaton* $\mathscr{A} = (A,\delta,\sigma,\tau)$, denoted as $[\![\mathscr{A}]\!]$, is a fuzzy language in $\mathscr{F}(X^*)$ defined by

$$[\![\mathscr{A}]\!](u) = \bigvee_{a,b\in A} \sigma(a) \otimes \delta^*(a,u,b) \otimes \tau(b), \tag{2.11}$$

for any $u = x_1x_2\ldots x_n \in X^+$, where $x_1,x_2,\ldots,x_n \in X$. In other words, the equality (2.11) means that the membership degree of the word $u$ to the fuzzy language $[\![\mathscr{A}]\!]$ is equal to the degree to which $\mathscr{A}$ recognizes or accepts the word $u$. Using notation from (1.42), and the second equality in (1.44), we can state (2.11) as

$$[\![\mathscr{A}]\!](u) = \sigma \circ \delta_u \circ \tau. \tag{2.12}$$

Fuzzy automata $\mathscr{A}$ and $\mathscr{B}$ are called *language equivalent*, or sometimes just *equivalent*, if $[\![\mathscr{A}]\!] = [\![\mathscr{B}]\!]$.

Fuzzy automata $\mathscr{A} = (A,\delta^A,\sigma^A,\tau^A)$ and $\mathscr{A}' = (A',\delta^{A'},\sigma^{A'},\tau^{A'})$ are *isomorphic* if there is a bijective function $\phi : A \to A'$ such that $\delta^A_x(a,b) = \delta^{A'}_x(\phi(a),\phi(b))$, for all $a,b \in A$ and $x \in X$, and also, $\sigma^A(a) = \sigma^{A'}(\phi(a))$ and $\tau^A(a) = \tau^{A'}(\phi(a))$, for every $a \in A$. It is easy to check that in this case we also have that

$$\delta^A_u(a,b) = \delta^{A'}_u(\phi(a),\phi(b)), \text{ for all } a,b \in A \text{ and } u \in X^*. \tag{2.13}$$

Given a fuzzy language $\varphi : X^* \to L$ and $v \in X^*$, we define the fuzzy language $v^{-1}\varphi : X^* \to L$ and $\varphi v^{-1} : X^* \to L$ by letting $v^{-1}\varphi(u) = \varphi(vu)$ and $\varphi v^{-1}(u) = \varphi(uv)$ for $u \in X^*$. The fuzzy language $v^{-1}\varphi$ is called a *left derivative*, and the fuzzy language $\varphi v^{-1}$ a *right derivative* of $\varphi$ with respect to $v$.

Cardinality of a fuzzy automaton $\mathscr{A} = (A,\delta,\sigma,\tau)$, in notation $|\mathscr{A}|$, is defined as the cardinality of its set of states $A$. A fuzzy automaton $\mathscr{A}$ is called *minimal fuzzy automaton* of a language $f \in \mathscr{F}(X^*)$ if it recognizes $f$ and $|\mathscr{A}| < |\mathscr{A}'|$, for any fuzzy automaton $\mathscr{A}'$ recognizing $f$. A minimal fuzzy automaton recognizing a given fuzzy language $f$ is not necessarily unique up to an isomorphism. This is also true for nondeterministic automata.

For more information on fuzzy automata over complete residuated lattices we refer to [33, 116, 53, 54, 55, 96, 100, 125, 128, 57].

## 2.3. Crisp-deterministic fuzzy (weighted) automata

Let $\mathscr{A} = (A, \delta, \sigma, \tau)$ be a fa over $X$ and $\mathscr{L}$, respectively a wa over $X$ and $K$. The fuzzy (weighted) transition function $\delta$ is called *crisp-deterministic* if for every $x \in X$ and every $a \in A$ there exists $a' \in A$ such that $\delta_x(a, a') = 1$, and $\delta_x(a, b) = 0$, for all $b \in A \setminus \{a'\}$. The fuzzy set of initial states (initial weight vector) $\sigma$ is called *crisp-deterministic* if there exists $a_0 \in A$ such that $\sigma(a_0) = 1$ and $\sigma(a) = 0$ for every $a \in A \setminus \{a_0\}$. If both $\sigma$ and $\delta$ are crisp-deterministic, then $\mathscr{A}$ is called a *crisp-deterministic fuzzy (weighted) finite automaton*, (for short: *cdffa* (*cdwfa*))

Equivalently, we can define a crisp-deterministic fuzzy (weighted) finite automaton over $X$ and $\mathscr{L}$ ($K$) as a quadruple $\mathscr{A} = (A, \delta, a_0, \tau)$, where $A$ is a non-empty *set of states*, $\delta : A \times X \to A$ is a *transition function*, $a_0 \in A$ is an *initial state* and $\tau : A \to L$ ($\tau : A \to K$) is a *fuzzy set of final states* (*final weight vector*). The transition function $\delta$ can be extended to a function $\delta^* : A \times X^* \to A$ as follows: $\delta^*(a, \varepsilon) = a$, for each $a \in A$, and $\delta^*(a, ux) = \delta(\delta^*(a, u), x)$, for every $a \in A$, $u \in X^*$ and $x \in X$. Also, we allow the set $A$ to be infinite, and then $\mathscr{A}$ is called a *crisp-deterministic fuzzy (weighted) automaton* (for short: *cdfa* (*cdwa*)). The *language*(*behaviour*) of $\mathscr{A}$ is the fuzzy language (series) $[\![\mathscr{A}]\!] : X^* \to L$ ($[\![\mathscr{A}]\!] \in K\langle\langle X^* \rangle\rangle$) defined by

$$([\![\mathscr{A}]\!], u) = \tau(\delta^*(a_0, u)) \,. \tag{2.14}$$

for every $u \in X^*$. Obviously, the image of $[\![\mathscr{A}]\!]$ is contained in the image of $\tau$ which is finite if the set of states $A$ is finite. By $|\mathscr{A}|$ we denote the cardinality of the set of states of $\mathscr{A}$, i.e., $|\mathscr{A}| = |A|$.

A fuzzy language (series) $\varphi : X^* \to L$ ($\varphi \in K\langle\langle X^* \rangle\rangle$) is called *cdffa-recognizable* (*cdfwa-recognizable*) if there exists a crisp-deterministic fuzzy (weighted) finite automaton $\mathscr{A}$ over $X$ and $\mathscr{L}$ ($K$) such that $[\![\mathscr{A}]\!] = \varphi$. We also say that $\mathscr{A}$ *recognizes* $\varphi$.

A crisp-deterministic fuzzy (weighted) automaton $\mathscr{A}$ is called *minimal crisp-deterministic fuzzy (weighted) automaton* of a language (series) $f \in \mathscr{F}(X^*)$ ($f \in K\langle\langle X^* \rangle\rangle$) if it recognizes $f$ and $|\mathscr{A}| < |\mathscr{A}'|$, for any crisp-deterministic fuzzy (weighted) automaton $\mathscr{A}'$ recognizing $f$.

For a fuzzy language $\varphi : X^* \to L$, let $A_\varphi = \{u^{-1}\varphi \mid u \in X^*\}$ denote the set of all left derivatives of $\varphi$, and let $\delta_\varphi : A_\varphi \times X \to A_\varphi$ and $\tau_\varphi : A_\varphi \to L$ be mappings defined by

$$\delta_\varphi(\psi, x) = x^{-1}\psi \quad \text{and} \quad \tau_\varphi(\psi) = \psi(\varepsilon), \tag{2.15}$$

for every $\psi \in A_\varphi$ and $x \in X$. Then $\mathscr{A}_\varphi = (A_\varphi, \delta_\varphi, \varphi, \tau_\varphi)$ is an accessible cdfa, and it is called the *derivative automaton* of the fuzzy language $\varphi$ [57, 53]. It has been proved in [57] that the derivative automaton $\mathscr{A}_\varphi$ is finite if and only if the fuzzy language $\varphi$ is cdffa-recognizable. Namely, $\mathscr{A}_\varphi$ is a minimal cdfa which recognizes $\varphi$ [57]. An algorithm for construction of the derivative automaton of a fuzzy language, based on simultaneous construction of the derivative automata of ordinary languages $\varphi^{-1}(a)$, for all $a \in \mathrm{Im}(\varphi)$, has been also given in [57].

Let $\mathscr{A} = (A, \delta, a_0, \tau)$ and $\mathscr{A}' = (A', \delta', a_0', \tau')$ be cdfa (cdwa). If $\phi : A \to A'$ is a mapping satisfying $\phi(a_0) = a_0'$, $\phi(\delta(a,x)) = \delta'(\phi(a), x)$, for all $a \in A$ and $x \in X$, and $\tau(a) = \tau'(\phi(a))$, for every $a \in A$, then $\phi$ is called a *homomorphism* of these cdfa (cdwa). If $\varphi$ is a bijective homomorphism, then it is called an *isomorphism* of cdfa (cdwa), and we say that $\mathscr{A}$ and $\mathscr{A}'$ are *isomorphic* cdfa (cdwa).

Let $\mathscr{A} = (A, \delta, a_0, \tau)$ be a cdfa (cdwa). An equivalence relation $\mu$ on $A$ is a *congruence* on $\mathscr{A}$ if $(a,b) \in \mu$ implies $\tau(a) = \tau(b)$ and $(\delta(a,x), \delta(b,x)) \in \mu$, for all $a, b \in A$ and $x \in X$ (cf. [57]). If $\mu$ is a congruence on $\mathscr{A}$ and $A_\mu = A/\mu$ is the related factor set, then we define mappings $\delta_\mu : A_\mu \times X \to A_\mu$ and $\tau_\mu : A_\mu \to L$ ($\tau_\mu : A_\mu \to K$) by $\delta_\mu([a]_\mu, x) = [\delta(a,x)]_\mu$ and $\tau_\mu([a]_\mu) = \tau(a)$, for all $a \in A$ and $x \in X$, and the quadruple $\mathscr{A}_\mu = (A_\mu, \delta_\mu, [a_0]_\mu, \tau_\mu)$ is a cdwa (cdfa), called the *factor automaton* of $\mathscr{A}$ with respect to $\mu$.

## 2.4. Afterset and factor fuzzy automata

For a fuzzy automaton $\mathscr{A} = (A, \delta^A, \sigma^A, \tau^A)$ over an alphabet $X$ and a complete residuated lattice $\mathscr{L}$, and a fuzzy quasi-order $R$ on $A$, the *afterset fuzzy automaton* of $\mathscr{A}$ with respect to $R$ is a fuzzy automaton $\mathscr{A}/R = (A/R, \delta^{A/R}, \sigma^{A/R}, \tau^{A/R})$ whose set of states is the set $A/R = \{R_a \mid a \in A\}$ of all $R$-aftersets, the fuzzy transition function $\delta^{A/R} : A/R \times X \times A/R \to L$ is defined by

$$\delta^{A/R}(R_a, x, R_b) = \bigvee_{a', b' \in A} R(a,a') \otimes \delta^A(a', x, b') \otimes R(b', b), \tag{2.16}$$

or equivalently by

$$\delta^{A/R}(R_a, x, R_b) = (R \circ \delta_x^A \circ R)(a,b) = R_a \circ \delta_x^A \circ R^b, \tag{2.17}$$

for all $R_a, R_b \in A/R$ and $x \in X$, the fuzzy set $\sigma^{A/R} : A/R \to L$ of initial states is defined by

$$\sigma^{A/R}(R_a) = \bigvee_{a' \in A} \sigma^A(a') \otimes R(a', a) = (\sigma^A \circ R)(a) = \sigma^A \circ R^a, \quad R_a \in A/R; \tag{2.18}$$

and the fuzzy set $\tau^{A/R} : A/R \to L$ of terminal states is defined by

$$\tau^{A/R}(R_a) = \bigvee_{a' \in A} R(a,a') \otimes \tau^A(a') = (R \circ \tau^A)(a) = R_a \circ \tau^A, \quad R_a \in A/R. \tag{2.19}$$

The fuzzy language $[\![\mathscr{A}/R]\!]$ recognized by the afterset fuzzy automaton $\mathscr{A}/R$ is given by

$$\begin{aligned} &[\![\mathscr{A}/R]\!](\varepsilon) = \sigma^A \circ R \circ \tau^A, \\ &[\![\mathscr{A}/R]\!](u) = \sigma^A \circ R \circ \delta_{x_1}^A \circ R \circ \delta_{x_2}^A \circ R \circ \cdots \circ R \circ \delta_{x_n}^A \circ R \circ \tau^A \end{aligned} \tag{2.20}$$

for every $u = x_1x_2\ldots x_n \in X^*$, where $x_1, x_2, \ldots, x_n \in X$.

Let us note that the previous equation follows directly from the definition of the afterset automaton $\mathscr{A}/R$ and the fact that $R \circ R = R$ for every fuzzy quasi-order $R$.

Analogously, for a fuzzy automaton $\mathscr{A} = (A, \delta^A, \sigma^A, \tau^A)$, the *foreset fuzzy automaton* of $\mathscr{A}$ w.r.t. $R$ is a fuzzy automaton $\mathscr{A}\backslash R = (A\backslash R, \delta^{A\backslash R}, \sigma^{A\backslash R}, \tau^{A\backslash R})$ defined as follows: the set of states is $A\backslash R = \{R^a \mid a \in A\}$, the fuzzy transition function $\delta^{A\backslash R}$ is defined by

$$\begin{aligned}\delta^{A\backslash R}(R^a, x, R^b) &= \bigvee_{a',b' \in A} R(a,a') \otimes \delta^A(a',x,b') \otimes R(b',b) \\ &= (R \circ \delta^A_x \circ R)(a,b) = R_a \circ \delta^A_x \circ R^b, \end{aligned} \tag{2.21}$$

for all $R^a, R^b \in A\backslash R$ and $x \in X$, the fuzzy set $\sigma^{A\backslash R} \in L^{A\backslash R}$ of initial states is given by

$$\sigma^{A\backslash R}(R^a) = \bigvee_{a' \in A} \sigma^A(a') \otimes R(a',a) = (\sigma^A \circ R)(a) = \sigma^A \circ R^a, \quad R^a \in A\backslash R, \tag{2.22}$$

and the fuzzy set $\tau^{A\backslash R} \in L^{A\backslash R}$ of terminal states is given by

$$\tau^{A\backslash R}(R^a) = \bigvee_{a' \in A} R(a,a') \otimes \tau^A(a') = (R \circ \tau^A)(a) = R_a \circ \tau^A, \quad R^a \in A\backslash R. \tag{2.23}$$

It can easily be shown that:

**Theorem 2.1.** *For any fuzzy quasi-order $R$ on a fuzzy automaton $\mathscr{A}$ the afterset fuzzy automaton $\mathscr{A}/R$ and the foreset fuzzy automaton $\mathscr{A} \setminus R$ are isomorphic.*

It is interesting to note that in the case when $R$ is a fuzzy equivalence relation on $A$, the afterset fuzzy automaton $\mathscr{A}/R$ and the foreset fuzzy automaton $\mathscr{A} \setminus R$ will be equal, and in this case the automaton $\mathscr{A}/R$ ($\mathscr{A} \setminus R$) will be called the *factor fuzzy automaton* of $\mathscr{A}$ with respect to $R$.

# Chapter 3
# Reduced Nerode automaton

Determinization of a weighted/fuzzy finite automaton is considered here as a procedure of its conversion into an equivalent crisp-deterministic weighted/fuzzy automaton, which is deterministic, but it has weights/truth values attached to terminal (final) states. This kind of determinism was first studied by Bělohlávek [12], for fuzzy finite automata over a complete distributive lattice, and Li and Pedrycz [72], for fuzzy finite automata over a lattice-ordered monoid, where such algorithms were given which generalize the subset construction. Another algorithm, provided by Ignjatović et al. [54], is also a generalization of the subset construction and for any input it generates a smaller crisp-deterministic fuzzy automaton than the algorithms from [12, 72]. Since this crisp-deterministic fuzzy automaton can be alternatively constructed by means of the Nerode right congruence of the original fuzzy finite automaton, it was called in [57] the Nerode automaton of this fuzzy finite automaton. The Nerode automaton was constructed in [54] for fuzzy finite automata over a complete residuated lattice, but it was noted that the same construction can also be applied to fuzzy finite automata over a lattice-ordered monoid, and moreover, to weighted finite automata over a semiring.

Recently, weighted automata with weights in more general structures have attracted the attention of researchers. These structures are called strong bimonoids, and can be viewed as semirings that might lack distributivity. Examples of strong bimonoids include the real unit interval [0,1] with t-conorm and t-norm from multi-valued logic [68], the algebraic cost structure from algebraic path problems [73], etc. Important natural examples of strong bimonoids are ortho-modular lattices, which serve as a basis of quantum logics where distributivity typically fails. Automata based on quantum logics, i.e., automata with weights in orthomodular lattices, have been investigated in [70, 98, 101, 130, 131, 132]. Automata modeling, e.g., peak power consumption of energy and with particular strong bimonoids as weight structures were recently studied in [23, 24, 25]. Fuzzy automata and fuzzy tree automata defined by a pair of a t-conorm and a t-norm on the real unit interval were investigated in [8, 9] respectively [10], and their study for non-distributive

pairs has been appraised as especially interesting. Automata over arbitrary bounded lattices were investigated in [38].

Study of weighted finite automata over arbitrary strong bimonoids has been recently initiated by Droste et al. [37], and methods and algorithms for their determinization have been studied by Ćirić et al. [27]. Due to lack of distributivity, there are three different definitions of the behavior of weighted finite automata over strong bimonoids: the run semantics, the initial algebra semantics, and the transition semantics. The Nerode automaton was also constructed for a weighted finite automaton over a strong bimonoid, and it was proved that it is equivalent to the original automaton with respect to the initial algebra semantics. With respect to transition and run semantics, crisp-deterministic weighted automata equivalent to the original weighted automaton are respectively the Myhill automaton and the run automaton (cf. [27]). Each of these three automata can have strictly less number of states than the other two automata. Namely, in [27] certain examples were given which show that finiteness of the Nerode and Myhill automaton, and definability of the run automaton, are completely independent of each other, i.e., all combinations of finiteness/infiniteness of the Nerode and Myhill automaton, and definability/undefinability of the run automaton are possible. However, in the case of semirings all three semantics coincide for every weighted finite automaton, and in this case the Nerode automaton has always smaller cardinality than the corresponding Myhill and run automaton.

The aim of this chapter is to provide an new algorithm for determinization of weighted finite automata over strong bimonoids which generates a crisp-deterministic weighted automaton that is equivalent to the original weighted automaton with respect to the initial algebra semantics, and is even smaller than the Nerode automaton. This automaton was called in [61] the *reduced Nerode automaton*. Therefore, in determinization of weighted finite automata over strong bimonoids and semirings, and fuzzy finite automata over lattice-ordered monoids, this algorithm gives smaller crisp-deterministic automata than any other known determinization algorithm for that kind of automata.

The organization of this chapter is the following. In Section 3.1., we recall the definition of the Nerode automaton of the given weghted finite automaton over a strong bimonoid. In Section 3.2., given any weighted finite automaton $\mathscr{A}$, we define the reduced Nerode automaton $\mathscr{A}_R$, and we show that it is a crisp-deterministic weighted automaton equivalent to $\mathscr{A}$ with respect to the initial algebra semantics (cf. Theorem 3.2). In Section 3.3. we prove that the cardinality of the reduced Nerode automaton $\mathscr{A}_R$ is always less or equal than the cardinality of the related Nerode automaton $\mathscr{A}_N$ (cf. Theorem 3.4), and we demonstrate by an example that this inequality may be strict (Example 3.9). In fact, we show that the automaton $\mathscr{A}_R$ can be obtained by reduction of the Nerode automaton $\mathscr{A}_N$ by means of a suitable congruence on $\mathscr{A}_N$. For that reason, $\mathscr{A}_R$ is called the reduced Nerode automaton of $\mathscr{A}$.

In contrast to nondeterministic finite automata, for which there is always an equivalent deterministic finite automaton, for weighted and fuzzy finite

automata there is no necessarily an equivalent crisp-deterministic finite automaton. In other words, determinization procedures for weighted and fuzzy finite automata may result in a crisp-deterministic automaton with possibly infinitely many states. Consequently, it is interesting to investigate the conditions under which these automata are finite. Here we provide necessary and sufficient conditions for the reduced Nerode automaton $\mathscr{A}_R$ to be finite (cf. Theorem 3.7). In the final section, we develop an algorithm which computes $\mathscr{A}_R$ if it is finite. We also prove minimality results for the reduced Nerode automaton $\mathscr{A}_R$ in a suitable class of crisp-deterministic weighted automata (Theorem 3.10).

The determinization algorithm provided here for weighted and fuzzy automata generalizes determinization using the transition sets, developed in [120, 121] for nondeterministic automata, and improves determinization algorithms for weighted and fuzzy finite automata provided in [12, 27, 54, 72].

## 3.1. Nerode automaton

Let $\mathscr{A} = (A, \delta, \sigma, \tau)$ be a weighted finite automaton over an alphabet $X$ and a strong bimonoid $K$. Set $A_N = \{\sigma_u \mid u \in X^*\}$, and define $\delta_N : A_N \times X \to A_N$ and $\tau_N : A_N \to K$ by

$$\delta_N(\sigma_u, x) = \sigma_{ux} \quad \text{and} \quad \tau_N(\sigma_u) = \sigma_u \cdot \tau, \tag{3.1}$$

for every $u \in X^*$ and $x \in X$. Then $\mathscr{A}_N = (A_N, \delta_N, \sigma_\varepsilon, \tau_N)$ is a crisp-deterministic weighted automaton over $X$ and $K$, and it is $i$-equivalent to $\mathscr{A}$, i.e., $[\![\mathscr{A}_N]\!] = [\![\mathscr{A}]\!]_i$ (cf. [27]).

The automaton $\mathscr{A}_N = (A_N, \delta_N, \sigma_\varepsilon, \tau_N)$ is called the *Nerode automaton* of the weighted finite automaton $\mathscr{A}$. The concept of the Nerode automaton was first introduced in [54] (see also [57]), and it was used there in determinization of fuzzy finite automata over complete residuated lattices. Moreover, it was noted in [54] that the same construction can be used for determinization of fuzzy finite automata over lattice-ordered monoids and weighted finite automata over semirings. Not long afterward, Nerode automata were also associated with weighted finite automata over strong bimonoids (cf. [27]). However, there are three different definitions of the behavior of such automata: the run semantics, the initial algebra semantics, and the transition semantics. The Nerode automaton $\mathscr{A}_N$ is equivalent to the original automaton $\mathscr{A}$ with respect to the initial algebra semantics, and with respect to the transition and run semantics, crisp-deterministic weighted automata equivalent to $\mathscr{A}$ are respectively the *Myhill automaton* $\mathscr{A}_M$ and the *run automaton* $\mathscr{A}_\pi$ (cf. [27, 57]). Generally, the cardinalities of the sets of states of these three automata are not comparable. In particular, in [27] certain examples were given which demonstrate that finiteness of $\mathscr{A}_N$ and $\mathscr{A}_M$, and definability of $\mathscr{A}_\pi$, are completely independent of each other, i.e., for a wfa $\mathscr{A}$ all combinations of

finiteness/infiniteness of $\mathscr{A}_N$ and $\mathscr{A}_M$, and definability/undefinability of $\mathscr{A}_\pi$ are possible. But, in the case of semirings all three semantics coincide for every weighted finite automaton, and in this case the Nerode automaton always has smaller cardinality than the corresponding Myhill automaton and run automaton. Here we construct a crisp-deterministic weighted automaton which always has smaller cardinality than the Nerode automaton.

## 3.2. Definition of the reduced Nerode automaton

In the rest of this chapter, if not noted otherwise, $K$ will be a strong bimonoid, and we will assume that $X = \{x_1, x_2, ..., x_m\}$, for some $m \in \mathbb{N}$.

Let $\mathscr{A} = (A, \delta, \sigma, \tau)$ be a weighted automaton over $X$ and $K$. For each word $u \in X^*$ and each letter $x \in X$ let us define a vector $\eta_u^x : A \to K$ by

$$\eta_u^x = \sigma_{ux}, \tag{3.2}$$

i.e., $\eta_u^x = \sigma_u \cdot \delta_x$. Moreover, for each word $u \in X^*$ we define a scalar $\kappa_u \in K$ by

$$\kappa_u = \sigma_u \cdot \tau. \tag{3.3}$$

Next, for each word $u \in X^*$ we define the *weighted transition* $(m+1)$*-tuple* $\theta_u$ by

$$\theta_u = (\eta_u^{x_1}, ..., \eta_u^{x_m}, \kappa_u). \tag{3.4}$$

The following lemma shows how the weighted transition $(m+1)$-tuple $\theta_{ux}$ can be computed from $\theta_u$, for arbitrary $u \in X^*$ and $x \in X$.

**Lemma 3.1.** *Let $\mathscr{A} = (A, \delta, \sigma, \tau)$ be a weighted finite automaton over $X$ and $K$. Then for all $u \in X^*$ and $x \in X$ the following is true:*

$$\theta_{ux} = (\eta_u^x \cdot \delta_{x_1}, ..., \eta_u^x \cdot \delta_{x_m}, \eta_u^x \cdot \tau). \tag{3.5}$$

*Proof.* According to (3.2) and (3.3), for each $j \in \{1, \ldots, m\}$ we have

$$\eta_{ux}^{x_j} = \sigma_{uxx_j} = \sigma_{ux} \cdot \delta_{x_j} = \eta_u^x \cdot \delta_{x_j}, \quad \kappa_{ux} = \sigma_{ux} \cdot \tau = \eta_u^x \cdot \tau,$$

and hence, (3.5) holds. □

Now, let us set $A_R = \{\theta_u \mid u \in X^*\}$, and let us define $\delta_R : A_R \times X \to A_R$ and $\tau_R : A_R \to K$ by

$$\delta_R(\theta_u, x) = \theta_{ux} \quad \text{and} \quad \tau_R(\theta_u) = \kappa_u, \tag{3.6}$$

for every $u \in X^*$ and $x \in X$.

We have that the following is true.

**Theorem 3.1.** *Let $\mathscr{A} = (A, \delta, \sigma, \tau)$ be a weighted finite automaton over $X$ and $K$. Then $\mathscr{A}_R = (A_R, \delta_R, \theta_\varepsilon, \tau_R)$ is a crisp-deterministic weighted automaton over $X$ and $K$, and it is $i$-equivalent to $\mathscr{A}$, i.e. $[\![\mathscr{A}_R]\!] = [\![\mathscr{A}]\!]_i$.*

*Proof.* Let $u, v \in X^*$ such that $\theta_u = \theta_v$, that is $\eta_u^x = \eta_v^x$ for all $x \in X$, and $\kappa_u = \kappa_v$. Then by Lemma 3.1, for each $x \in X$ we have that

$$\theta_{ux} = (\eta_u^x \cdot \delta_{x_1}, \ldots, \eta_u^x \cdot \delta_{x_m}, \eta_u^x \cdot \tau) = (\eta_v^x \cdot \delta_{x_1}, \ldots, \eta_v^x \cdot \delta_{x_m}, \eta_v^x \cdot \tau) = \theta_{vx},$$

and therefore, $\delta_R$ is a well-defined mapping. It is clear that $\tau_R$ is also a well-defined mapping. By this it follows that $\mathscr{A}_R = (A_R, \delta_R, \theta_\varepsilon, \tau_R)$ is a crisp-deterministic weighted automaton.

Finally, let us show that $\mathscr{A}$ is $i$-equivalent to $\mathscr{A}_R$. For any $u \in X^*$ we have that

$$([\![\mathscr{A}_R]\!], u) = \tau_R(\delta_R(\theta_\varepsilon, u)) = \tau_R(\theta_u) = \kappa_u = \sigma_u \cdot \tau = ([\![\mathscr{A}]\!]_i, u),$$

and therefore $[\![\mathscr{A}_R]\!] = [\![\mathscr{A}]\!]_i$. □

Note that for every word $u \in X^*$ the weighted transition $(m+1)$-tuple $\theta_u = (\eta_u^{x_1}, \ldots, \eta_u^{x_m}, \kappa_u)$ can be viewed as the ordered pair whose first component is the $n$-tuple $(\eta_u^{x_1}, \ldots, \eta_u^{x_m})$, and the second component is $\kappa_u$. On the other hand, the $n$-tuple $(\eta_u^{x_1}, \ldots, \eta_u^{x_m})$ can be identified with the $X \times A$-matrix $\eta_u$ over $K$ defined by

$$\eta_u(x, a) = \eta_u^x(a), \tag{3.7}$$

for all $x \in X$ and $a \in A$, since for arbitrary $u, v \in X^*$ we have that

$$\begin{aligned} \eta_u = \eta_v &\Leftrightarrow (\forall x \in X)(\forall a \in A)\ \eta_u(x, a) = \eta_v(x, a) \Leftrightarrow \\ &\Leftrightarrow (\forall x \in X)(\forall a \in A)\ \eta_u^x(a) = \eta_v^x(a) \Leftrightarrow \\ &\Leftrightarrow (\forall x \in X)\ \eta_u^x = \eta_v^x \Leftrightarrow (\eta_u^{x_1}, \ldots, \eta_u^{x_m}) = (\eta_v^{x_1}, \ldots, \eta_v^{x_m}). \end{aligned}$$

Therefore, $\theta_u$ can be considered as the pair $(\eta_u, \kappa_u)$.

According to (3.2) and (3.7), for all $u \in X^*$, $x \in X$ and $a \in A$ we have that

$$\eta_u(x, a) = \sigma_{ux}(a) = (\sigma_u \cdot \delta_x)(a) = \sum_{b \in A} \sigma_u(b) \cdot \delta_x(b, a). \tag{3.8}$$

In particular, in the case when $\mathscr{A}$ is a nondeterministic (Boolean) automaton, $\eta_u$ can be naturally interpreted as follows: *$\eta_u$ is the set of all outgoing transitions from the set of states $\sigma_u$* (cf. [120, 121]). In this case, unlike the subset construction, which deals with sets of states, the construction of $\mathscr{A}_R$ deals with sets of transitions.

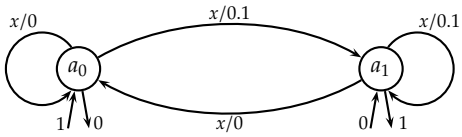



**Fig. 3.1** The transition graph of the wfa $\mathscr{A}$ from Example 4.1

## 3.3. Finiteness conditions

It is well-known that the Nerode automaton of a weighted finite automaton over a strong bimonoid is not necessarily finite (cf. [27, 54]). This also holds for the automaton $\mathscr{A}_R$, as the following example shows.

*Example 3.1.* Let $\mathscr{A} = (A, \delta, \sigma, \tau)$ be a weighted finite automaton over the alphabet $X = \{x\}$ and the Viterbi semiring $K$ given by the transition graph shown in Figure 3.1. Then for every $u \in X^*$ we have that

$$\eta_u^x(a) = \begin{cases} 0, & \text{if } a = a_0, \\ 0.1^{|u|+1}, & \text{otherwise,} \end{cases} \qquad \kappa_u = 0.1^{|u|},$$

where $|u|$ denotes the length of the word $u$. Therefore, there are infinitely many pairs of the form $\theta_u = (\eta_u^x, \kappa_u)$, so the set $A_R$ is infinite.

Now we show that the automaton $\mathscr{A}_R$ always has smaller or equal number of states than the corresponding Nerode automaton $\mathscr{A}_N$.

**Theorem 3.2.** *Let $\mathscr{A} = (A, \delta, \sigma, \tau)$ be a weighted finite automaton over $X$ and $K$. Then $|\mathscr{A}_R| \leqslant |\mathscr{A}_N|$.*

*Proof.* Let us define a mapping $\phi : A_N \to A_R$ by

$$\phi(\sigma_u) = \theta_u,$$

for every $u \in X^*$. Let $u, v \in X^*$ such that $\sigma_u = \sigma_v$. Then

$$\phi(\sigma_u) = \theta_u = (\sigma_u \cdot \delta_{x_1}, \ldots, \sigma_u \cdot \delta_{x_n}, \sigma_u \cdot \tau) = (\sigma_v \cdot \delta_{x_1}, \ldots, \sigma_v \cdot \delta_{x_n}, \sigma_v \cdot \tau) = \theta_v = \phi(\sigma_v),$$

so we have that $\phi$ is well-defined mapping. It is clear that $\phi$ is surjective, and hence, $|\mathscr{A}_R| \leqslant |\mathscr{A}_N|$. □

We will show later that $\mathscr{A}_R$ may have strictly smaller number of states than $\mathscr{A}_N$ (cf. Example 3.2).

It is easy to verify that $\phi$ is a homomorphism of crisp-deterministic weighted automata, and its kernel $\varrho = \ker \phi$ is a congruence on the Nerode automaton $\mathscr{A}_N$ given by

$$(\sigma_u,\sigma_v)\in\varrho \;\Leftrightarrow\; \Big((\forall x\in X)\,\delta_N(\sigma_u,x)=\delta_N(\sigma_v,x)\Big)\;\&\;\tau_N(\sigma_u)=\tau_N(\sigma_v)$$
$$\Leftrightarrow\; \Big((\forall w\in X^+)\,\delta_N^*(\sigma_u,w)=\delta_N^*(\sigma_v,w)\Big)\;\&\;\tau_N(\sigma_u)=\tau_N(\sigma_v),$$

for all $u,v\in X^*$. Hence, the automaton $\mathscr{A}_R$ can be obtained by reduction of the Nerode automaton $\mathscr{A}_N$ by means of the congruence $\varrho$. For that reason we will call $\mathscr{A}_R$ the *reduced Nerode automaton* of $\mathscr{A}$.

Remind that the greatest congruence $\gamma$ on $\mathscr{A}_N$ is given by

$$(\sigma_u,\sigma_v)\in\gamma \;\Leftrightarrow\; (\forall w\in X^*)\,\tau_N\Big(\delta_N^*(\sigma_u,w)\Big)=\tau_N\Big(\delta_N^*(\sigma_v,w)\Big),$$

for all $u,v\in X^*$, and the related factor automaton $\mathscr{A}_N/\gamma$ is minimal (cf. [57]). The congruence $\varrho$ resembles the greatest congruence $\gamma$, but in general it is different from it, i.e., the automaton $\mathscr{A}_R$ is not necessarily minimal (cf. Example 3.2).

Let $S$ be semigroup and $\mu$ a crisp equivalence on $S$. If for all $a,b,x\in S$, we have that $(a,b)\in\mu$ implies $(ax,bx)\in\mu$, then $\mu$ is called a *right congruence* on $S$. Next, let $\mu$ be a right congruence on the free monoid $X^*$, and let $[u]_\mu$ denote the equivalence class of $u\in X^*$ with respect to $\mu$. Then one can define a function $\delta_\mu:(X^*/\mu)\times X\to(X^*/\mu)$ by

$$\delta_\mu([u]_\mu,x)=[ux]_\mu,$$

for all $u\in X^*$ and $x\in X$, and for every series $\varphi\in K\langle\langle X^*\rangle\rangle$ such that $\mu\subseteq\ker\varphi$ one can define a function $\tau_\mu^\varphi:(X^*/\mu)\to K$ by

$$\tau_\mu^\varphi([u]_\mu)=(\varphi,u),$$

for each $u\in X^*$. Now, $\mathscr{A}_\mu=(X^*/\mu,\delta_\mu,[\varepsilon]_\mu,\tau_\mu^\varphi)$ is a crisp-deterministic weighted automaton over $X$ and $K$ such that $[\![\mathscr{A}_\mu]\!]=\varphi$, and it is called the *right congruence automaton* associated with $\mu$ (cf. [57]).

If $\mathscr{A}=(A,\delta,\sigma,\tau)$ is a weighted finite automaton over $X$ and $K$, then the relation $N_{\mathscr{A}}$ on $X^*$ defined by

$$(u,v)\in N_{\mathscr{A}} \;\Leftrightarrow\; \sigma_u=\sigma_v,$$

for all $u,v\in X^*$, is a right congruence on $X^*$, and it is known as the *Nerode right congruence* of $\mathscr{A}$. We can regard the Nerode automaton $\mathscr{A}_N$ as the right congruence automaton associated with $N_{\mathscr{A}}$, and just for that reason we called $\mathscr{A}_N$ the Nerode automaton. Similarly, a relation $R_{\mathscr{A}}$ on $X^*$ defined by

$$(u,v)\in R_{\mathscr{A}} \;\Leftrightarrow\; \theta_u=\theta_v,$$

for all $u,v\in X^*$, is also a right congruence on $X^*$, and we can regard the crisp-deterministic weighted automaton $\mathscr{A}_R$ as the right congruence automaton associated with $R_{\mathscr{A}}$.

Let $\mathscr{A} = (A, \delta, \sigma, \tau)$ be a weighted finite automaton over $X$ and $K$. For every $\tau' : A \to K$, the weighted finite automaton $(A, \delta, \sigma, \tau')$ is called a *final variant* of $\mathscr{A}$, and by $\mathrm{Rec}_i(\mathscr{A}, \sigma)$ we denote the family of all series in $K\langle\langle X^* \rangle\rangle$ which are $i$-behaviors of some final variant of $\mathscr{A}$. For each $a \in A$, a series $[\![\mathscr{A}]\!]^{(\sigma,a)}$ in $K\langle\langle X^* \rangle\rangle$ is defined by

$$([\![\mathscr{A}]\!]^{(\sigma,a)}, u) = \sigma_u(a),$$

for every $u \in X^*$. The following result was proved in [27].

**Theorem 3.3.** *Let $\mathscr{A} = (A, \delta, \sigma, \tau)$ be a weighted finite automaton over $X$ and $K$. Then the following conditions are equivalent:*

(1) *The Nerode automaton $\mathscr{A}_N$ is finite.*

(2) *The Nerode right congruence $N_{\mathscr{A}}$ of $\mathscr{A}$ has finite index.*

(3) *The series $[\![\mathscr{A}]\!]^{(\sigma,a)}$ has a finite image, for every $a \in A$.*

(4) *Any series from $\mathrm{Rec}_i(\mathscr{A}, \sigma)$ is cdwfa-recognizable.*

Remind that the *index* of an equivalence is defined as the cardinality of the related factor set, i.e., an equivalence relation has *finite index* if it has finitely many equivalence classes.

Further, for any weighted finite automaton $\mathscr{A} = (A, \delta, \sigma, \tau)$ over $X$ and $K$, and any $x \in X$ and $a \in A$, let us define a series $[\![\mathscr{A}]\!]^{(\eta,x,a)}$ in $K\langle\langle X^* \rangle\rangle$ by

$$([\![\mathscr{A}]\!]^{(\eta,x,a)}, u) = \eta_u^x(a)$$

for each $u \in X^*$. It is easy to verify that $[\![\mathscr{A}]\!]^{(\eta,x,a)} = [\![\mathscr{A}_R^{(x,a)}]\!]$, where $\mathscr{A}_R^{(x,a)} = (A_R, \delta_R, \theta_\varepsilon, \tau_R^{(x,a)})$ is the final variant of $\mathscr{A}_R$ with the final weight vector $\tau_R^{(x,a)}$ defined by $\tau_R^{(x,a)}(\theta_u) = \eta_u^x(a)$, for every $u \in X^*$. Also, for every $a \in A$, $x \in X$ and $u \in X^*$ we have that

$$([\![\mathscr{A}]\!]^{(\eta,x,a)}, u) = \eta_u^x(a) = \sigma_{ux}(a) = ([\![\mathscr{A}]\!]^{(\sigma,a)}, ux),$$

and hence, the series $[\![\mathscr{A}]\!]^{(\eta,x,a)}$ is the right derivate of the series $[\![\mathscr{A}]\!]^{(\sigma,a)}$ with respect to $x$.

**Lemma 3.2.** *Let $\mathscr{A} = (A, \delta, \sigma, \tau)$ be a weighted finite automaton over $X$ and $K$, and let $x \in X$. Then the right derivative of an arbitrary series from $\mathrm{Rec}_i(\mathscr{A}, \sigma)$ with respect to $x$ is the behavior of some final variant of $\mathscr{A}_R$.*

*Proof.* Let $\varphi$ be an arbitrary series from $\mathrm{Rec}_i(\mathscr{A}, \sigma)$. According to the definition of $\mathrm{Rec}_i(\mathscr{A}, \sigma)$, there exists a final variant $\mathscr{A}' = (A, \delta, \sigma, \tau')$ of $\mathscr{A}$ such that $[\![\mathscr{A}']\!]_i = \varphi$, i.e,

$$(\varphi, u) = ([\![\mathscr{A}']\!]_i, u) = \sigma_u \cdot \tau',$$

for every $u \in X^*$. Therefore,

$$(\varphi x^{-1}, u) = (\varphi, ux) = \sigma_{ux} \cdot \tau',$$

for every $u \in X^*$. Let us now consider the final variant $\mathscr{A}_R^x = (A_R, \delta_R, \theta_\varepsilon, \tau_R^x)$ of $\mathscr{A}_R$, where $\tau_R^x(\theta_u) = \eta_u^x \cdot \tau'$. Then we have that

$$([\![\mathscr{A}_R^x]\!], u) = \tau_R^x(\delta_R(\theta_\varepsilon, u)) = \tau_R^x(\theta_u) = \eta_u^x \cdot \tau' = \sigma_{ux} \cdot \tau' = (\varphi x^{-1}, u),$$

for all $u \in X^*$, and hence, $[\![\mathscr{A}_R^x]\!] = \varphi x^{-1}$. □

The following theorem determines necessary and sufficient conditions under which the reduced Nerode automaton is finite.

**Theorem 3.4.** *Let $\mathscr{A} = (A, \delta, \sigma, \tau)$ be a weighted finite automaton over $X$ and $K$. Then the following conditions are equivalent:*

(1) *The reduced Nerode automaton $\mathscr{A}_R$ is finite;*

(2) *The Nerode automaton $\mathscr{A}_N$ is finite;*

(3) *The right congruence $R_{\mathscr{A}}$ has finite index;*

(4) *The series $[\![\mathscr{A}]\!]^{(\eta,x,a)}$, for all $x \in X$ and $a \in A$, and $[\![\mathscr{A}]\!]_i$ have finite images;*

(5) *The series $\varphi x^{-1}$, for all $\varphi \in \mathrm{Rec}_i(\mathscr{A}, \sigma)$ and $x \in X$, and $[\![\mathscr{A}]\!]_i$ are cdwfa-recognizable.*

*Proof.* (1)⇒(2). Let $A_R = \{\theta_{u_1}, \ldots, \theta_{u_k}\}$, for some $k \in \mathbb{N}$ and $u_1, \ldots, u_k \in X^*$. For every $u \in X^+$ we have that $u = vx$, for some $v \in X^*$ and $x \in X$, and $\theta_v = \theta_{u_j}$, for some $j \in \{1, \ldots, k\}$, so $\sigma_u = \sigma_{vx} = \eta_v^x = \eta_{u_j}^x = \sigma_{u_j x}$. Hence,

$$A_N \setminus \{\sigma_\varepsilon\} \subseteq \{\sigma_{u_j x_l} \mid 1 \leqslant j \leqslant k, 1 \leqslant l \leqslant m\},$$

and therefore, $|A_N| \leqslant km + 1$, i.e., $|\mathscr{A}_N| \leqslant |\mathscr{A}_R| \cdot |X| + 1$.

(2)⇒(1). This follows immediately by Theorem 3.2.

(1)⇔(3). By definition of the right congruence $R_{\mathscr{A}}$, the rule $\theta_u \mapsto [u]_{T_\theta}$, for each $u \in X^*$, defines a bijective mapping of $A_R$ onto $X^*/R_{\mathscr{A}}$. By this it follows that (1)⇔(3) holds.

(1)⇒(5). According to Theorem 3.1 we have that $[\![\mathscr{A}]\!]_i = [\![\mathscr{A}_R]\!]$, and hence, $[\![\mathscr{A}]\!]_i$ is cdwfa-recognizable. Also, for any series $\varphi \in \mathrm{Rec}_i(\mathscr{A}, \sigma)$ and any $x \in X$, by Lemma 3.2 it follows that the right derivative $\varphi x^{-1}$ is the behavior of some final variant of $\mathscr{A}_R$, and consequently, it is cdwfa-recognizable.

(5)⇒(4). By the assumption (5), $[\![\mathscr{A}]\!]_i$ is cdwfa-recognizable, and therefore, it has a finite image. Further, for all $a \in A$ and $x \in X$ we have that $[\![\mathscr{A}]\!]^{(\eta,x,a)} = [\![\mathscr{A}]\!]^{(\sigma,a)} x^{-1}$, and $[\![\mathscr{A}]\!]^{(\sigma,a)} \in \mathrm{Rec}_i(\mathscr{A}, \sigma)$. According to (v), we obtain that $[\![\mathscr{A}]\!]^{(\eta,x,a)}$ is cdwfa-recognizable, and hence, it has a finite image.

(4)⇒(1). For each $u \in X^*$, $x \in X$ and $a \in A$ we have $\eta_u^x(a) = ([\![\mathscr{A}]\!]^{(\eta,x,a)}, u)$ and $\kappa_u = ([\![\mathscr{A}]\!]_i, u)$. Thus, the number of different $(m+1)$-tuples $\theta_u = (\eta_u^{x_1}, \ldots, \eta_u^{x_m}, \kappa_u)$ $(u \in X^*)$ is bounded by

$$|\mathrm{Im}([\![\mathscr{A}]\!]_i)| \cdot \prod_{x \in X, a \in A} |\mathrm{Im}([\![\mathscr{A}]\!]^{(\eta,x,a)})|,$$

which is finite by assumption. □

## 3.4. Algorithm for computing the reduced Nerode automaton

Now, we give the following algorithm for computing the reduced Nerode automaton.

**Algorithm 3.5 (*Construction of the reduced Nerode automaton* $\mathscr{A}_R$)** The input of this algorithm is a weighted finite automaton $\mathscr{A} = (A, \delta, \sigma, \tau)$ over an alphabet $X = \{x_1, ..., x_m\}$ and $K$, and the output is the cdwa $\mathscr{A}_R = (A_R, \delta_R, \theta_\varepsilon, \tau_R)$ which is $i$-equivalent to $\mathscr{A}$.

The procedure is to construct the *transition tree* of $\mathscr{A}_R$ directly from $\mathscr{A}$. It is constructed inductively in the following way:

(A1) The root of the tree is $\theta_\varepsilon = (\sigma \circ \delta_{x_1}, ..., \sigma \circ \delta_{x_m}, \sigma \circ \tau)$, and we put $T_0 = \{\theta_\varepsilon\}$.

(A2) After the $i$-th step let a tree $T_i$ have been constructed, and vertices in $T_i$ have been labelled either 'closed' or 'non-closed'. The meaning of these two terms will be made clear in the sequel.

(A3) In the next step we construct a tree $T_{i+1}$ by enriching the tree $T_i$ as follows: for every non-closed leaf $\theta_u = (\eta_u^{x_1}, ..., \eta_u^{x_n}, \kappa_u)$ occurring in $T_i$, where $u \in X^*$, and every $x \in X$, we add a vertex $\theta_{ux} = (\eta_u^x \cdot \delta_{x_1}, ..., \eta_u^x \cdot \delta_{x_m}, \eta_u^x \cdot \tau)$ and an edge from $\theta_u$ to $\theta_{ux}$ labelled by $x$. If, in addition, $\theta_{ux}$ is a state that has already been constructed, then we mark $\theta_{ux}$ as *closed*. The procedure terminates when all leaves are marked closed.

(A4) Simultaneously, for each non-closed leaf $\theta_u$ occurring in $T_i$, where $u \in X^*$, we compute the value $\tau_R(\theta_u)$ using the formula (3.6).

(A5) When the transition tree of $\mathscr{A}_R$ is constructed, we erase all closure marks and glue leaves to interior vertices with the same label. The diagram that results is the transition graph of $\mathscr{A}_R$.

If the semiring $K$ is weakly locally finite, then the algorithm terminates in a finite number of steps, for any wfa over $K$, and the result is a cdwfa.

On the other hand, if $K$ is not locally finite, then the algorithm terminates in a finite number of steps under conditions determined by Theorem 3.4.

The next example demonstrates application of the above algorithm, and shows that the reduced Nerode automaton may be strictly smaller than the corresponding Nerode automaton and it is not necessarily minimal.

*Example 3.2.* Let $\mathscr{A} = (A, \delta, \sigma, \tau)$ be a weighted finite automaton over the alphabet $X = \{x\}$ and the Boolean semiring $\mathbb{B}$ given by the transition graph shown in Figure 6.1.

Then $\sigma$, $\delta_x$ and $\tau$ are represented by

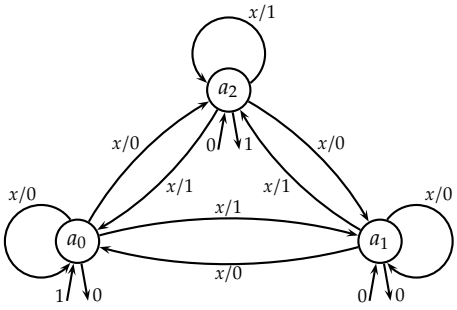



**Fig. 3.2** The transition graph of the wfa $\mathscr{A}$ from Example 3.2

$$\sigma = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}, \quad \delta_x = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 1 \end{bmatrix}, \quad \tau = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix},$$

and we have that

$$\eta^x_\varepsilon = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix}, \quad \kappa_\varepsilon = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = 0,$$

$$\eta^x_x = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}, \quad \kappa_x = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = 0,$$

$$\eta^x_{x^2} = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 \end{bmatrix}, \quad \kappa_{x^2} = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = 1,$$

$$\eta^x_{x^3} = \begin{bmatrix} 1 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}, \quad \kappa_{x^3} = \begin{bmatrix} 1 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = 1,$$

$$\eta^x_{x^4} = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} = \eta^x_{x^3}, \quad \kappa_{x^4} = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = 1 = \kappa_{x^3}.$$

Hence, for all $u \in X^*$ such that $|u| \geqslant 3$ (where $|u|$ is the length of the word $u$) we have that

$$\eta^x_u = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}, \quad \kappa_u = 1.$$

Therefore $A_R = \{\theta_\varepsilon, \theta_x, \theta_{x^2}, \theta_{x^3}\}$, and the transition tree and the graph of the cdwfa $\mathscr{A}_R = (A_R, \delta_R, \theta_\varepsilon, \tau_R)$ are shown in Figure 6.2.

Note that the series $[\![\mathscr{A}_R]\!] = [\![\mathscr{A}]\!]_l$ is given by

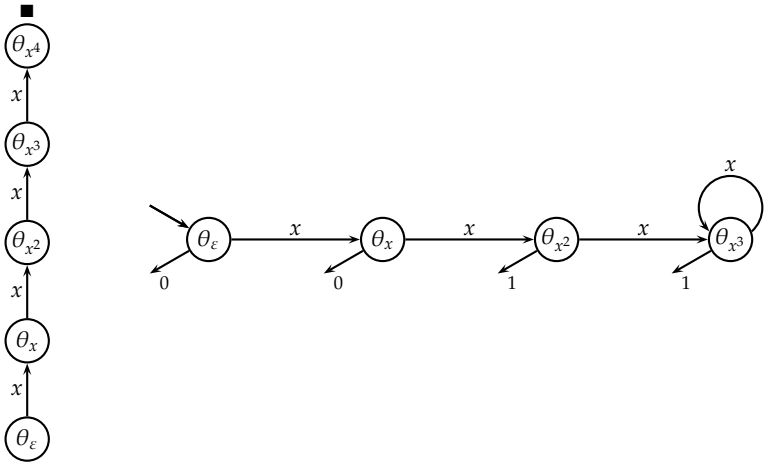



**Fig. 3.3** The transition tree and the graph of the automaton $\mathscr{A}_R$ of the wfa $\mathscr{A}$ given in Figure 6.1.

$$([\![\mathscr{A}_R]\!], u) = ([\![\mathscr{A}]\!]_i, u) = \begin{cases} 0, & \text{if } u = \varepsilon \text{ or } u = x, \\ 1, & \text{otherwise,} \end{cases}$$

for all $u \in X^*$. The Nerode automaton $\mathscr{A}_N$ of $\mathscr{A}$ is shown in Figure 3.4 (left). Clearly, $\mathscr{A}_N$ has more states than $\mathscr{A}_R$, and $\mathscr{A}_R$ is obtained from $\mathscr{A}_N$ by merging states $\sigma_{x^3}$ and $\sigma_{x^4}$. The reduced Nerode automaton $\mathscr{A}_R$ *is not minimal*. The minimal cdwa of the series $[\![\mathscr{A}_R]\!] = [\![\mathscr{A}_N]\!] = [\![\mathscr{A}]\!]_i$ is shown in Figure 3.4 (right). It can be obtained from $\mathscr{A}_R$ by merging states $\theta_{x^2}$ and $\theta_{x^3}$.

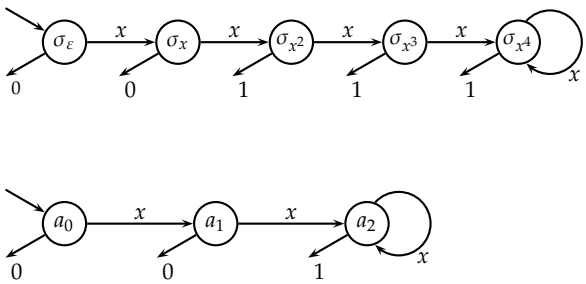



**Fig. 3.4** The transition graph of the Nerode automaton $\mathscr{A}_N$ (up) of the wfa $\mathscr{A}$ given in Figure 6.1, and the minimal cdwa (down) of the series $[\![\mathscr{A}_R]\!] = [\![\mathscr{A}_N]\!] = [\![\mathscr{A}]\!]_i$.

According to the proof of Theorem 3.4, if the alphabet $X$ is a singleton, then the Nerode automaton $\mathscr{A}_N$ can have at most one state more than the reduced Nerode automaton $\mathscr{A}_R$, but for larger alphabets this difference in the number of states between $\mathscr{A}_N$ and $\mathscr{A}_R$ may be much greater.

Although the reduced Nerode automaton $\mathscr{A}_R$ is not necessarily a minimal cdwa for the series $[\![\mathscr{A}]\!]_i$, we show that it is a minimal cdwa for some families of series including $[\![\mathscr{A}]\!]_i$.

**Theorem 3.6.** *Let $\mathscr{A} = (A, \delta, \sigma, \tau)$ be a weighted finite automaton over $X$ and $K$. Then the following holds:*

(a) *The automaton $\mathscr{A}_R$ is minimal among all crisp-deterministic weighted automata whose final variants recognize $[\![\mathscr{A}]\!]_i$ and all series $[\![\mathscr{A}]\!]^{(\eta,x,a)}$, where $x \in X$ and $a \in A$;*

(b) *The automaton $\mathscr{A}_R$ is minimal among all crisp-deterministic weighted automata whose final variants recognize $[\![\mathscr{A}]\!]_i$ and all series $\varphi x^{-1}$, where $x \in X$ and $\varphi \in \mathrm{Rec}_i(\mathscr{A}, \sigma)$.*

*Proof.* (a) As we have already shown, $[\![\mathscr{A}]\!]_i$ is the behavior of $\mathscr{A}_R$, and for all $x \in X$ and $a \in A$, $[\![\mathscr{A}]\!]^{(\eta,x,a)}$ is the behavior of $\mathscr{A}_R^{(x,a)}$, which is a final variant of $\mathscr{A}_R$. Let $\mathscr{A}' = (A', \delta', a'_0, \tau')$ be an arbitrary cdwa such that each series $[\![\mathscr{A}]\!]^{(\eta,x,a)}$, for $x \in X$ and $a \in A$, as well as $[\![\mathscr{A}]\!]_i$, is recognized by some final variant of $\mathscr{A}'$. Then for all $x \in X$ and $a \in A$ there exists $\tau'_{(x,a)} : A' \to K$ such that

$$\tau'_{(x,a)}(\delta'^*(a'_0, u)) = ([\![\mathscr{A}]\!]^{(\eta,x,a)}, u) = \eta^x_u(a),$$

for every $u \in X^*$, and there exists $\tau'' : A' \to K$ such that

$$\tau''(\delta'^*(a'_0, u)) = ([\![\mathscr{A}]\!]_i, u) = \kappa_u,$$

for every $u \in X^*$. We may assume that $\mathscr{A}'$ is accessible.

Define a mapping $\psi : A' \longrightarrow A_R$ by setting $\psi(a') = \theta_u$, where $a' \in A'$ and $u \in X^*$ such that $a' = \delta'^*(a'_0, u)$. To show that $\psi$ is well-defined, consider $a' \in A'$ and $u, v \in X^*$ such that $a' = \delta'^*(a'_0, u) = \delta'^*(a'_0, v)$. Then for all $x \in X$ and $a \in A$ we have that

$$\eta^x_u(a) = \tau'_{(x,a)}(\delta'^*(a'_0, u)) = \tau'_{(x,a)}(\delta'^*(a'_0, v)) = \eta^x_v(a),$$
$$\kappa_u = \tau''(\delta'^*(a'_0, u)) = \tau''(\delta'^*(a'_0, v)) = \kappa_v,$$

and hence $\theta_u = \theta_v$. It is easy to show that $\psi$ is surjective, and therefore $|\mathscr{A}_R| \leqslant |\mathscr{A}'|$.

(b) This assertion is an immediate consequence of Lemma 3.2, the assertion (a), and the fact that for all $x \in X$ and $a \in A$, $[\![\mathscr{A}]\!]^{(\eta,x,a)}$ is the right derivative of $[\![\mathscr{A}]\!]^{(\sigma,a)} \in \mathrm{Rec}_i(\mathscr{A}, \sigma)$ with respect to $x$. □

# Chapter 4
# Brzozowski type determinization for fuzzy automata

Way back in 1963 year, J. A. Brzozowski [7] invented an elegant algorithm for determinization of nondeterministic finite automata. When its input is a nondeterministic finite automaton, the algorithm alternates two reversion and determinization procedures and produces a minimal deterministic finite automaton equivalent to the starting automaton. Hence, the algorithm performs both determinization and minimization. In addition, when the input is a deterministic finite automaton, the algorithm performs its minimization by doing just one reversion and determinization procedure. The worst-case complexity of Brzozowski's algorithm is exponential, seeing that there are regular languages such that the minimal deterministic automaton of its reverse language is exponentially larger than the minimal deterministic automaton of this language. However, Brzozowski's algorithm frequently performs better than this worst case would suggest and it outperforms theoretically faster algorithms.

The purpose of this chapter is to adapt the Brzozowski's double reversal algorithm to fuzzy automata. We start from an arbitrary fuzzy automaton and we show that performing the construction of a reverse Nerode automaton twice we obtain a minimal crisp-deterministic fuzzy automaton equivalent to the original fuzzy automaton. We also demonstrate by example that the fuzzy version of the Brzozowski's algorithm outperforms all previous methods for determinization of fuzzy automata developed by Bělohlávek [12], Li and Pedrycz [72], Ignjatović et al. [54], and Jančić et al. [61], in the sense that it not only produces a smaller automaton than all these methods, but even when all these methods produce infinite automata, the Brzozowski type determinization can produce a finite one. Moreover, when the starting fuzzy automaton is crisp-deterministic and accessible, its minimization is performed applying just one construction of the reverse Nerode automaton

The chapter is organized as follows. In the first section we introduce the notions of the reverse Nerode automaton and the reverse fuzzy language, and show that the reverse Nerode automaton of a given fuzzy automaton $\mathscr{A}$

recognize the reverse fuzzy language of the fuzzy language recognized by $\mathscr{A}$. In Section 4.2. we first introduce the concept of the right fuzzy language associated with a state of a fuzzy automaton and describe some basic properties of right languages. After that, we construct the right language automaton of a fuzzy automaton $\mathscr{A}$, and prove that it is isomorphic to the derivative automaton of the fuzzy language recognized by $\mathscr{A}$ (Theorem 4.1), and consequently, if all right fuzzy languages associated with states of an accessible crisp-deterministic fuzzy automaton $\mathscr{A}$ are pairwise different, we show that $\mathscr{A}$ is minimal. Then we prove that the reverse Nerode automaton of any accessible crisp-deterministic fuzzy automaton $\mathscr{A}$ is a minimal crisp-deterministic fuzzy automaton equivalent to the reverse automaton of $\mathscr{A}$ (Theorem 4.2).

Further, in Section 4.3. we define the concept of a Brzozowski automaton of a fuzzy automaton $\mathscr{A}$ and prove that it is a minimal crisp-deterministic fuzzy automaton equivalent to $\mathscr{A}$ (Theorem 4.3). Finally, we give a simple example of a fuzzy automaton $\mathscr{A}$ for which all previously known determinization methods produce an infinite crisp-deterministic fuzzy automaton, while Brzozowski automaton of $\mathscr{A}$ is finite and has only three states.

In the final section, we develop algorithm for constructing a Brzozowski automaton of a given fuzzy automaton $\mathscr{A}$. It is important to note that the same determinization and minimization method, without any modifications, can be applied to weighted automata over a commutative semiring.

## 4.1. Reverse Nerode automaton

Let $\mathscr{A} = (A, \delta, \sigma, \tau)$ be a fuzzy automaton over $\mathscr{L}$ and $X$. The *reverse fuzzy automaton* of $\mathscr{A}$ is a fuzzy automaton $\overline{\mathscr{A}} = (A, \bar{\delta}, \bar{\sigma}, \bar{\tau})$, where $\bar{\sigma} = \tau$, $\bar{\tau} = \sigma$, and $\bar{\delta} : A \times X \times A \to L$ is defined by:

$$\bar{\delta}(a, x, b) = \delta(b, x, a),$$

for all $a, b \in A$ and $x \in X$. Roughly speaking, the reverse automaton of $\mathscr{A}$ is obtained from $\mathscr{A}$ by exchanging fuzzy sets of initial and final states and "reversing" all the transitions.

Due to the fact that the multiplication $\otimes$ is commutative, we have that $\bar{\delta}_u(a, b) = \delta_{\bar{u}}(b, a)$, for all $a, b \in A$ and $u \in X^*$.

For a fuzzy language $f \in \mathscr{F}(X^*)$, the *reverse fuzzy language* of $f$ is a fuzzy language $\overline{f} \in \mathscr{F}(X^*)$ defined by $\overline{f}(u) = f(\bar{u})$, for each $u \in X^*$. As $\overline{(\bar{u})} = u$ for all $u \in X^*$, we have that $\overline{(\overline{f})} = f$, for each fuzzy language $f$.

If $\mathscr{A}$ is a fuzzy automaton over $\mathscr{L}$ and $X$, it is easy to see that the reverse fuzzy automaton $\overline{\mathscr{A}}$ recognizes the reverse fuzzy language $\overline{[\![\mathscr{A}]\!]}$ of the fuzzy language $[\![\mathscr{A}]\!]$ recognized by $\mathscr{A}$, i.e., $[\![\overline{\mathscr{A}}]\!] = \overline{[\![\mathscr{A}]\!]}$.

Let $\mathscr{A} = (A, \delta, \sigma, \tau)$ be a fuzzy automaton over $X$ and $\mathscr{L}$. For each $u \in X^*$ we define fuzzy sets $\sigma_u, \tau_u \in \mathscr{F}(A)$ as follows:

$$\sigma_u(a) = \bigvee_{b \in A} \sigma(b) \otimes \delta^*(b, u, a), \quad \tau_u(a) = \bigvee_{b \in A} \delta^*(b, u, a) \otimes \tau(b),$$

for each $a \in A$. Equivalently,

$$\sigma_u = \sigma \circ \delta_u, \qquad \tau_u = \delta_u \circ \tau.$$

As we have mentioned before, the Nerode automaton of $\mathscr{A} = (A, \delta, \sigma, \tau)$ is the crisp-deterministic automaton $\mathscr{A}_N = (A_N, \delta_N, \sigma_\varepsilon, \tau_N)$ whose set of states is $A_N = \{\sigma_u \mid u \in X^*\}$, and functions $\delta_N : A_N \times X \longrightarrow A_N$ and $\tau_N : A_N \to L$ are defined by

$$\delta_N(\sigma_u, x) = \sigma_{ux}, \qquad \tau_N(\sigma_u) = \sigma_u \circ \tau, \tag{4.1}$$

for all $u \in X^*$ and $x \in X$. It was also shown that the Nerode automaton of a fuzzy automaton $\mathscr{A}$ over a complete residuated lattice is a crisp-deterministic fuzzy automaton equivalent to $\mathscr{A}$, i.e., $[\![\mathscr{A}_N]\!] = [\![\mathscr{A}]\!]$.

By the *reverse Nerode automaton* of $\mathscr{A}$ we will mean the Nerode automaton of the reverse fuzzy automaton $\overline{\mathscr{A}}$ of $\mathscr{A}$. For the sake of simplicity, we denote the reverse Nerode automaton of $\mathscr{A}$ by $\mathscr{A}_{\overline{N}}$ (instead of $(\overline{\mathscr{A}})_N$). Let us note that $\mathscr{A}_{\overline{N}} = (A_{\overline{N}}, \delta_{\overline{N}}, \tau_\varepsilon, \tau_{\overline{N}})$, where $A_{\overline{N}} = \{\tau_u \mid u \in X^*\}$, and the functions $\delta_{\overline{N}} : A_{\overline{N}} \times X \to A_{\overline{N}}$ and $\tau_{\overline{N}} : A_{\overline{N}} \to L$ are given by

$$\delta_{\overline{N}}(\tau_u, x) = \tau_{xu}, \qquad \tau_{\overline{N}}(\tau_u) = \tau_u \circ \sigma, \tag{4.2}$$

for all $u \in X^*$ and $x \in X$.

## 4.2. Fuzzy languages associated with states

Let $\mathscr{A} = (A, \delta, \sigma, \tau)$ be a fuzzy automaton over $X$ and $\mathscr{L}$.

For any state $a \in A$, the *right fuzzy language associated with a* is the fuzzy language $\tau_a \in \mathscr{F}(X^*)$ defined by

$$\tau_a(u) = \bigvee_{b \in A} \delta^*(a, u, b) \otimes \tau(b),$$

for each $u \in X^*$. In other words, $\tau_a$ is the fuzzy language recognized by a fuzzy automaton $\mathscr{A}' = (A, \delta, a, \tau)$ obtained from $\mathscr{A}$ by replacing $\sigma$ with the single crisp initial state $a$. The *left fuzzy language associated with a* is the fuzzy language $\sigma_a \in \mathscr{F}(X^*)$ given by

$$\sigma_a(u) = \bigvee_{b \in A} \sigma(b) \otimes \delta^*(b, u, a),$$

for each $u \in X^*$, i.e., the fuzzy language recognized by a fuzzy automaton $\mathscr{A}' = (A, \delta, \sigma, \{a\})$ obtained from $\mathscr{A}$ by replacing $\tau$ with the single crisp terminal state $a$.

We can easily show that the following is true.

**Lemma 4.1.** *The right fuzzy language associated with the state a of a fuzzy automaton $\mathscr{A}$ is equal to the reverse of the left fuzzy language associated with the state a in the reverse fuzzy automaton $\overline{\mathscr{A}}$.*

For a crisp-deterministic fuzzy automaton $\mathscr{A} = (A, \delta, a_0, \tau)$, the right fuzzy language associated with a state $a$ in $\mathscr{A}$ is given by

$$\tau_a(u) = \tau(\delta^*(a, u)), \tag{4.3}$$

for each $u \in X^*$, and in particular, $\tau_{a_0} = [\![\mathscr{A}]\!]$, i.e., the right fuzzy language associated with the initial state $a_0$ is the fuzzy language recognized by $\mathscr{A}$. It can be also easily verified that the following is true.

**Lemma 4.2.** *Let $\mathscr{A} = (A, \delta, a_0, \tau)$ be a crisp-deterministic fuzzy automaton. Then*

$$\tau_{\delta^*(a,u)} = u^{-1}\tau_a, \tag{4.4}$$

*for all $a \in A$ and $u \in X^*$.*

If $\mathscr{A} = (A, \delta, a_0, \tau)$ is a crisp-deterministic fuzzy automaton, we define another crisp-deterministic fuzzy automaton $\mathscr{A}_r = (A_r, \delta_r, \tau_{a_0}, \tau_r)$ as follows: the set of states $A_r$ is the set of all right fuzzy languages associated with states of $\mathscr{A}$, and $\delta_r : A_r \times X \to A_r$ and $\tau_r : A_r \to L$ are given by:

$$\delta_r(\tau_a, x) = \tau_{\delta(a,x)}, \qquad \tau_r(\tau_a) = \tau_a(\varepsilon),$$

for each $\tau_a \in A_r$. We have the following:

**Theorem 4.1.** *Let $\mathscr{A} = (A, \delta, a_0, \tau)$ be an accessible crisp-deterministic fuzzy automaton. Then $\mathscr{A}_r$ is an accessible crisp-deterministic fuzzy automaton isomorphic to the derivative automaton $\mathscr{A}_f$ of the fuzzy language $f = [\![\mathscr{A}]\!]$.*

*Proof.* Define a mapping $\phi : A_f \to A_r$ by

$$\phi(u^{-1}f) = \tau_{\delta^*(a_0,u)},$$

for each $u \in X^*$. If $u, v \in X^*$ such that $u^{-1}f = v^{-1}f$, then according to (4.4) we obtain that $\tau_{\delta^*(a_0,u)} = \tau_{\delta^*(a_0,v)}$, and hence $\phi(u^{-1}f) = \phi(v^{-1}f)$. Thus, $\phi$ is well-defined. On the other hand, let $u, v \in X^*$ such that $\phi(u^{-1}f) = \phi(v^{-1}f)$, i.e., $\tau_{\delta^*(a_0,u)} = \tau_{\delta^*(a_0,v)}$. Then by (4.4) it follows that

$$u^{-1}f = u^{-1}\tau_{a_0} = \tau_{\delta^*(a_0,u)} = \tau_{\delta^*(a_0,v)} = v^{-1}\tau_{a_0} = v^{-1}f.$$

Therefore, $\phi$ is injective. Due to the fact that $\mathscr{A}$ is accessible, it is easy to show that $\phi$ is a surjective mapping.

In order to prove that $\phi$ is a homomorphism, consider arbitrary $u \in X^*$ and $x \in X$. Then

$$\phi(\delta_f(u^{-1}f, x)) = \phi((ux)^{-1}f) = \tau_{\delta^*(a_0,ux)} = \\ = \tau_{\delta(\delta^*(a_0,u),x)} = \delta_r(\tau_{\delta^*(a_0,u)}, x) = \delta_r(\phi(u^{-1}f), x).$$

Moreover, $\phi(\varepsilon^{-1}f) = \tau_{a_0}$ and $\tau_f(u^{-1}f) = \tau_r(\phi(u^{-1}f))$. Hence, $\phi$ is an isomorphism. □

The automaton $\mathscr{A}_r$ will be called the *right language automaton* of $\mathscr{A}$.

By the previous theorem we obtain the following consequence.

**Corollary 4.1.** *Let $\mathscr{A} = (A, \delta, a_0, \tau)$ be an accessible crisp-deterministic fuzzy automaton. If all right fuzzy languages associated with states of $\mathscr{A}$ are pairwise different, then $\mathscr{A}$ is minimal.*

*Proof.* It is clear that the function $\phi : A \to A_r$ defined by $\phi(a) = \tau_a$ is a homomorphism of $\mathscr{A}$ onto $\mathscr{A}_r$. Therefore, if all right fuzzy languages associated with states of $\mathscr{A}$ are pairwise different, then $\phi$ is an isomorphism of $\mathscr{A}$ onto $\mathscr{A}_r$, and according to Theorem 4.1, $\mathscr{A}$ is minimal. □

Let us note that if $\mathscr{A} = (A, \delta, a_0, \tau)$ is a crisp-deterministic fuzzy automaton, then its reverse Nerode automaton is $\mathscr{A}_{\overline{N}} = (A_{\overline{N}}, \delta_{\overline{N}}, \tau_e, \tau_{\overline{N}})$, where $A_{\overline{N}}$ and $\delta_{\overline{N}} : A_{\overline{N}} \times X \to A_{\overline{N}}$ have the same form as in the general case, whereas the function $\tau_{\overline{N}} : A_{\overline{N}} \to L$ is given by

$$\tau_{\overline{N}}(\tau_u) = \tau_u(a_0), \tag{4.5}$$

for each $u \in X^*$.

Now, we are ready to prove the following

**Theorem 4.2.** *For an arbitrary accessible crisp-deterministic fuzzy automaton $\mathscr{A} = (A, \delta, a_0, \tau)$, the reverse Nerode automaton $\mathscr{A}_{\overline{N}}$ is a minimal crisp-deterministic fuzzy automaton equivalent to $\overline{\mathscr{A}}$.*

*Proof.* We have already noted that $\mathscr{A}_{\overline{N}}$ is a crisp-deterministic fuzzy automaton equivalent to $\overline{\mathscr{A}}$, so it remains to show that it is minimal. According to Corollary 4.1, it is enough to prove that all right fuzzy languages associated with states of $\mathscr{A}_{\overline{N}}$ are pairwise different.

Let $\tau_u, \tau_v \in A_{\overline{N}}$, where $u, v \in X^*$, be two different states of $\mathscr{A}_{\overline{N}}$. Then there is $a \in A$ such that $\tau_u(a) \neq \tau_v(a)$, and since $\mathscr{A}$ is accessible, there is $w \in X^*$ such that $a = \delta^*(a_0, w)$. According to (4.3) we obtain that

$$\tau_{\tau_u}(\overline{w}) = \tau_{\overline{N}}(\delta^*_{\overline{N}}(\tau_u, \overline{w})) = \tau_{\overline{N}}(\tau_{wu}) = \tau_{wu}(a_0) = \tau_{a_0}(wu) = \\ = \tau(\delta^*(a_0, wu)) = \tau(\delta(\delta^*(a_0, w), u)) = \tau(\delta^*(a, u)) = \tau_u(a),$$

whence $\tau_{\tau_u}(\overline{w}) = \tau_u(a)$, and in the same way we show that $\tau_{\tau_v}(\overline{w}) = \tau_v(a)$. Since $\tau_u(a) \neq \tau_v(a)$, we conclude that $\tau_{\tau_u}(\overline{w}) \neq \tau_{\tau_v}(\overline{w})$, and hence, $\tau_{\tau_u}$ and $\tau_{\tau_v}$ are different right fuzzy languages associated with states of $\mathscr{A}_{\overline{N}}$. □

## 4.3. Brzozowski automaton

Let $\mathscr{A}$ be a fuzzy automaton over an alphabet $X$ and a complete residuated lattice $\mathscr{L}$. The *Brzozowski automaton* of $\mathscr{A}$, in notation $\mathscr{A}_B$, is a fuzzy automaton obtained from $\mathscr{A}$ applying twice the construction of the reverse Nerode automaton, i.e.,

$$\mathscr{A}_B = \left(\mathscr{A}_{\overline{N}}\right)_{\overline{N}} = \left(\overline{(\overline{\mathscr{A}})_N}\right)_N.$$

Now we are ready to state and prove the main result of this chapter.

**Theorem 4.3.** *Let $\mathscr{A}$ be a fuzzy automaton over an alphabet $X$ and a complete residuated lattice $\mathscr{L}$. The Brzozowski automaton $\mathscr{A}_B$ is a minimal crisp-deterministic fuzzy automaton equivalent to $\mathscr{A}$.*

*Proof.* This follows immediately by Theorem 4.1 of [54] and Theorem 4.2.

Namely, according to Theorem 4.1 of [54], the reverse Nerode automaton $\mathscr{A}_{\overline{N}} = (\overline{\mathscr{A}})_N$ is a crisp-deterministic fuzzy automaton equivalent to $\overline{\mathscr{A}}$, and its reverse Nerode automaton $\left(\mathscr{A}_{\overline{N}}\right)_{\overline{N}}$ is a crisp-deterministic fuzzy automaton equivalent to $\mathscr{A}$. Therefore, by Theorem 4.2, $\mathscr{A}_B$ is a minimal crisp-deterministic fuzzy automaton equivalent to $\mathscr{A}$. □

Finally, we give the following example.

*Example 4.1.* Let $\mathscr{A} = (A, \delta, \sigma, \tau)$ be a fuzzy finite automaton over the alphabet $X = \{x\}$ and the Goguen (product) structure, given by the transition graph shown in Figure 1.

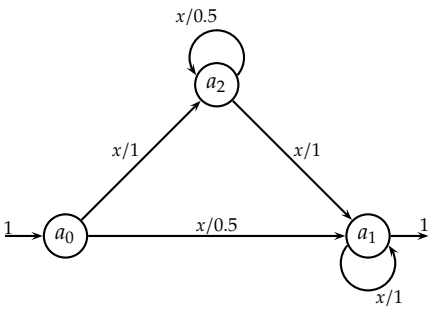



Figure 1. The transition graph of the fuzzy automaton $\mathscr{A}$

In the matrix form, $\sigma$, $\delta_x$ and $\tau$ are represented as follows:

$$\sigma = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}, \quad \delta_x = \begin{bmatrix} 0 & 0.5 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 0.5 \end{bmatrix}, \quad \tau = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}.$$

It easy to verify that $\sigma_x = \begin{bmatrix} 0 & 0.5 & 1 \end{bmatrix}$ and

$$\sigma_{x^n} = \begin{bmatrix} 0 & 1 & 0.5^{n-1} \end{bmatrix},$$

for each $n \in \mathbb{N}$, $n \geqslant 2$, which means that the Nerode automaton of $\mathscr{A}$ has infinitely many states.

On the other hand, it is not hard to check that the reverse Nerode automaton $\mathscr{A}_{\overline{N}}$ and the Brzozowski automaton $\mathscr{A}_B$ are mutually isomorphic, and they are represented by the graph in Figure 2.

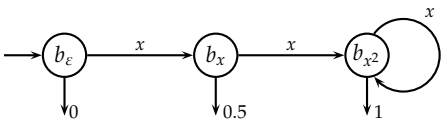



Figure 2. The transition graph of the reverse Nerode automaton $\mathscr{A}_{\overline{N}}$ and the Brzozowski automaton $\mathscr{A}_B$ of $\mathscr{A}$

The previous example demonstrates that there is a fuzzy finite automaton $\mathscr{A}$ whose Nerode automaton $\mathscr{A}_N$ is infinite, but the Brzozowski automaton $\mathscr{A}_B$ is finite. Note that the determinization methods developed by Bělohlávek [12] and Li and Pedrycz [72] always result in automata whose cardinality is greater than or equal to the cardinality of the related Nerode automaton (cf. [54]), and therefore, in this case these methods also give infinite automata. On the other hand, the method developed by Jančić et al. [61] always results in an automaton whose cardinality is less than or equal to the cardinality of the related Nerode automaton, but according to Theorem 3.7 of [61], this automaton is finite if and only if the related Nerode automaton is finite. Hence, in this case the method from [61] also gives an infinite automaton. Summing up, we conclude that all the above mentioned methods applied to the fuzzy finite automaton $\mathscr{A}$ from the previous example produce infinite automata, but Brzozowski automaton $\mathscr{A}_B$ is finite.

## 4.4. Algorithm for computing the Brzozowski automaton

As it was stated in the previous chapter, Brzozowski type algorithm for fuzzy automata is a procedure which construct a crisp-deterministic automaton $\mathscr{A}_B$ from a fuzzy automaton $\mathscr{A}$ applying twice the construction of the reverse Nerode automaton. For that reason, instead of giving an algorithm for constructing Brzozowski's automaton, we give an algorithm for constructing the reverse Nerode automaton, and in order to construct Brzozowski's automaton of a fuzzy automaton $\mathscr{A}$, one need to apply that algorithm twice.

**Algorithm 4.4 (*Construction of the reverse Nerode automaton* $\mathscr{A}_{\overline{N}}$)** The input of this algorithm is a fuzzy finite automaton $\mathscr{A} = (A, \delta, \sigma, \tau)$, and the output is the crisp-deterministic fuzzy automaton $\mathscr{A}_{\overline{N}} = (A_{\overline{N}}, \delta_{\overline{N}}, \tau_\varepsilon, \tau_{\overline{N}})$.

The procedure is to construct the *transition tree* of $\mathscr{A}_{\overline{N}}$ directly from $\mathscr{A}$, and during this procedure we use pointers $s(\cdot)$ which points vertices of the tree under construction to the corresponding integers. The transition tree of $\mathscr{A}_{\overline{N}}$ is constructed inductively as follows:

(A1) The root of the tree is $\tau_\varepsilon = \tau$, and we put $T_0 = \{\tau_\varepsilon\}$ and $s(\tau_\varepsilon) = 1$, and we compute the value $\tau_{\overline{N}}(\tau_\varepsilon) = \sigma \circ \tau_\varepsilon$.

(A2) After the $i$th step let a tree $T_i$ have been constructed, and vertices in $T_i$ have been labelled either 'closed' or 'non-closed'. The meaning of these two terms will be made clear in the sequel.

(A3) In the next step we construct a tree $T_{i+1}$ by enriching $T_i$ in the following way: for any non-closed leaf $\tau_u$ occuring in $T_i$, where $u \in X^*$, and any $x \in X$ we add a vertex $\tau_{xu} = \delta_x \circ \tau_u$ and an edge from $\tau_u$ to $\tau_{xu}$ labelled by $x$. Simultaneously, we check whether $\tau_{xu}$ is a fuzzy set that has already been constructed. If it is true, if $\tau_{xu}$ is equal to some previously computed $\tau_v$, we mark $\tau_{xu}$ as closed and set $s(\tau_{xu}) = s(\tau_v)$. Otherwise, we compute the value $\tau_{\overline{N}}(\tau_{xu}) = \sigma \circ \tau_{xu}$ and set $s(\tau_{xu})$ to be the next unassigned integer. The procedure terminates when all leaves are marked closed.

(A4) When the transition tree of $\mathscr{A}_{\overline{N}}$ is constructed, we erase all closure marks and glue leaves to interior vertices with the same pointer value. The diagram that results is the transition graph of $\mathscr{A}_{\overline{N}}$.

This procedure does not necessarily terminates in a finite number of steps, since the collection $\{\tau_u\}_{u \in X^*}$ may be infinite. However, in cases when this collection is finite, the procedure will terminate in a finite number of steps, after computing all its members.

Let $\mathscr{A} = (A, \delta, \sigma, \tau)$ be a fuzzy finite automaton with $n$ states and $m$ input letters, and suppose that the subsemiring $\mathscr{L}^*(\delta, \tau)$ of the semiring $(L, \vee, \otimes, 0, 1)$ generated by all membership values taken by $\delta$ and $\tau$ is finite and has $k$ elements. The algorithm builds the *transition tree* of the reverse Nerode automaton of $\mathscr{A}$, which is an $m$-ary tree with at most $k^n$ internal vertices that correspond to states of the crisp-deterministic fuzzy automaton under construction. Computationally most demanding part of the algorithm is the one where for any newly-constructed fuzzy set (a vertex of the tree) we check whether it has already been computed before. The computational time of this part, and the whole algorithm, is $O(mnk^{2n})$ (see the analysis of the computational time of Algorithm 5.15).

Before we perform the analysis of the computational time of the Brzozowski type algorithm for fuzzy finite automata, we formally state this algorithm.

**Algorithm 4.5 (*Construction of the Brzozowski automaton* $\mathscr{A}_B$)** The input of this algorithm is a fuzzy finite automaton $\mathscr{A} = (A, \delta, \sigma, \tau)$, and the output is the Brzozowski automaton $\mathscr{A}_B$.

The procedure consists of two steps:

(A1) We compute the reverse Nerode automaton $\mathscr{A}_{\overline{N}}$ of $\mathscr{A}$, by means of Algorithm 4.5.

(A2) Then we compute the reverse Nerode automaton of the automaton $\mathscr{A}_{\overline{N}}$. This reverse Nerode automaton of $\mathscr{A}_{\overline{N}}$ is the Brzozowski automaton $\mathscr{A}_B$

Let $\mathscr{A} = (A, \delta, \sigma, \tau)$ be a fuzzy finite automaton with $n$ states and $m$ input letters, and suppose that the subsemiring $\mathscr{L}^*(\delta, \sigma, \tau)$ of the semiring $(L, \vee, \otimes, 0, 1)$ generated by all membership values taken by $\delta$, $\sigma$ and $\tau$ is finite and has $k$ elements. As we have seen, the step (A1) of the previous algorithm produces an automaton with at most $k^n$ states, and the computational time of this step is $O(mnk^{2n})$. The second step may start from a much larger automaton, but despite this, this step produces a minimal crisp-deterministic fuzzy automaton equivalent to $\mathscr{A}$, an automaton not greater than the Nerode automaton of $\mathscr{A}$, which can not have more than $k^n$ states. Thus, again, the resulting transition tree can not have more than $k^n$ internal vertices, and consequently, step (A2) has the same computational time $O(mnk^{2n})$. This means that the total computational time of the Brzozowski's double reversal algorithm for the fuzzy automaton $\mathscr{A}$ is $O(mnk^{2n})$, the same as for constructions of the Nerode and the reverse Nerode automaton of $\mathscr{A}$.

No matter that Brzozowski type algorithm has been developed here for fuzzy automata over a complete residuated lattice, without any modification it can be also applied to fuzzy automata over lattice-ordered monoids and to weighted automata over commutative semirings.

# Chapter 5
# Simultaneous determinization and state reduction

In addition to the determinization, practical applications of automata often require the state reduction, a procedure of converting a given automaton into an equivalent automaton with a smaller number of states. This is a consequence of the fact that determinization algorithms may lead to an exponential growth in the number of states of an automaton. Due to the computational hardness of the state minimization problem for fuzzy finite automata, as well as for nondeterministic ones [79, 64, 133]), it is not required that this equivalent automaton is minimal, but it is necessary that it is effectively computable.

The state reduction problem for fuzzy automata was studied from different aspects in [2, 26, 77, 87, 91, 93, 94, 125, 129], as well as in the books [86, 92]. All algorithms provided there were motivated by the basic idea used in the minimization of ordinary deterministic automata, the idea of detecting and merging indistinguishable states, which comes down to computation of certain crisp equivalences on the set of states. A new approach to the state reduction was initiated in [33, 95]. First, it was shown that better reductions of fuzzy finite automata can be achieved if fuzzy equivalences are used instead of ordinary equivalences, and even better if fuzzy quasi-orders are used. In addition, it was shown that the state reduction problem for fuzzy finite automata can be reduced to the problem of finding fuzzy quasi-orders that are solutions to a particular system of fuzzy relation equations, called the general system. As the general system is difficult to solve, the problem was further reduced to the search for instances of the general system and their solutions which ensure the best possible reductions and can be efficiently computed. Two such instances, whose solutions were called right and left invariant, have the greatest solutions that can be computed in a polynomial time, and two other, whose solutions were called weakly right and left invariant, have the greatest solutions that ensure better reductions, but their computation requires an exponential time.

The main aim of this chapter is to combine determinization and state reduction methods into two-in-one algorithms that simultaneously perform determinization and state reduction. These algorithms perform better than all previous determinization algorithms for fuzzy finite automata, developed in [12, 72, 54, 61], in the sense that they produce smaller automata, while require the same computation time. The only exception is the Brzozowski type determinization algorithm, presented in the previous section, which produces a minimal crisp-deterministic fuzzy automaton, but we will see that the algorithms created here can be used within the Brzozowski type algorithm and improve its performance.

The chapter is organized as follows. In the first section we recall definitions of right and left invariant fuzzy relations, as well as definitons of weakly right and left invariant fuzzy relations on a given fuzzy automaton $\mathscr{A}$. In Section 5.2., for any fuzzy finite automaton $\mathscr{A}$ and a reflexive weakly right invariant fuzzy relation $\varphi$ on $\mathscr{A}$, we construct a crisp-deterministic fuzzy automaton $\mathscr{A}_\varphi$ and prove that it is equivalent to $\mathscr{A}$ (Theorem 5.1). If $\varphi$ is a weakly right invariant fuzzy quasi-order, we show that the same automaton $\mathscr{A}_\varphi$ would be produced if we first perform the state reduction of $\mathscr{A}$ by means of $\varphi$, and then we construct the Nerode automaton of this reduced automaton (Theorem 5.2). Furthermore, we show that automata $\mathscr{A}_\varphi$ determined by right invariant fuzzy quasi-orders are smaller than the Nerode automaton of $\mathscr{A}$, and that larger right invariant fuzzy quasi-orders determine smaller crisp-deterministic fuzzy automata. For a fuzzy finite automaton $\mathscr{A}$ and a reflexive weakly right invariant fuzzy relation $\varphi$ on $\mathscr{A}$, we also introduce the concept of the children automaton of $\mathscr{A}_\varphi$ and prove that it is equivalent to $\mathscr{A}$. In addition, if $\varphi$ is a right invariant fuzzy quasi-order, we prove that the children automaton of $\mathscr{A}_\varphi$ is smaller than $\mathscr{A}_\varphi$, and that larger right invariant fuzzy quasi-orders determine smaller children automata. We also show that weakly left invariant fuzzy quasi-orders play a completely different role in the determinization. Namely, if they are used to reduce the number of states prior the construction of the Nerode automaton, they will be unsuccessful because the Nerode automaton of the reduced fuzzy automaton would be the same as the Nerode automaton of the original one. However, we show that weakly left invariant fuzzy quasi-orders may be successful in combination with the construction of the reverse Nerode automaton and that a two-in-one algorithm can be provided which can improve performances of the Brzozowski type algorithm for fuzzy finite automata. In Section 5.3. we provide algorithms and perform the analysis of their computation time, and In Section 5.3. give characteristic computational examples.

## 5.1. Right and left invariant fuzzy relations

Let $\mathscr{A} = (A, \delta, \sigma, \tau)$ be a fuzzy automaton. A fuzzy relation $\varphi$ on $A$ is called *right invariant* if

$$\varphi \circ \delta_x \leqslant \delta_x \circ \varphi, \qquad \text{for each } x \in X, \tag{5.1}$$

$$\varphi \circ \tau \leqslant \tau, \tag{5.2}$$

and it is called *weakly right invariant* if

$$\varphi \circ \tau_u \leqslant \tau_u, \qquad \text{for each } u \in X^*. \tag{5.3}$$

Similarly we define the dual concepts. A fuzzy relation $\varphi$ on $A$ is called *left invariant* if

$$\delta_x \circ \varphi \leqslant \varphi \circ \delta_x, \qquad \text{for each } x \in X, \tag{5.4}$$

$$\sigma \circ \varphi \leqslant \sigma, \tag{5.5}$$

and it is called *weakly left invariant* if

$$\sigma_u \circ \varphi \leqslant \sigma_u, \qquad \text{for each } u \in X^*. \tag{5.6}$$

It is easy to verify that every right invariant fuzzy relation is weakly right invariant, and every left invariant fuzzy relation is weakly left invariant. Note that if $\varphi$ is reflexive, then $\varphi$ satisfies (5.3) if and only if it satisfies

$$\varphi \circ \tau_u = \tau_u, \qquad \text{for each } u \in X^*, \tag{5.7}$$

and $\varphi$ satisfies (5.6) if and only if it satisfies

$$\sigma_u \circ \varphi = \sigma_u, \qquad \text{for each } u \in X^*, \tag{5.8}$$

and consequently, $\varphi$ satisfies (5.2) if and only if it satisfies $\varphi \circ \tau = \tau$, and it satisfies (5.5) if and only if it satisfies $\sigma \circ \varphi = \sigma$. Moreover, if $\varphi$ is a fuzzy quasi-order, then $\varphi$ satisfies (5.1) if and only if it satisfies

$$\varphi \circ \delta^{Ax} \circ \varphi = \delta_x \circ \varphi, \qquad \text{for each } x \in X, \tag{5.9}$$

and $\varphi$ satisfies (5.4) if and only if it satisfies

$$\varphi \circ \delta_x \circ \varphi = \varphi \circ \delta_x, \qquad \text{for each } x \in X. \tag{5.10}$$

Right and left invariant fuzzy quasi-orders and fuzzy equivalences were introduced in [33, 116], where they were used in the state reduction of fuzzy automata. They are closely related to forward and backward simulations and bisimulations between fuzzy automata, which were studied in [30, 31]. Namely, a fuzzy quasi-order $\varphi$ on a fuzzy automaton $\mathscr{A}$ is right invariant if

its reverse $\varphi^{-1}$ is a forward simulation of $\mathscr{A}$ into itself, and $\varphi$ is left invariant if $\varphi$ is a backward simulation of $\mathscr{A}$ into itself.

Weakly right and left invariant fuzzy quasi-orders were introduced in [116], and they were also used in the state reduction of fuzzy automata, where they provide smaller automata than right invariant fuzzy quasi-orders, but are more difficult to compute. Weakly right and left invariant fuzzy quasi-orders and fuzzy equivalences are closely related to weak forward and backward simulations and bisimulations, which were studied in [59].

Algorithms for computing the greatest right and left invariant fuzzy quasi-orders on a fuzzy finite automaton, as well as algorithms for computing the greatest weakly right and left invariant ones, were provided in [116]. They are presented here in Section 5.3., together with an analysis of their computational time.

## 5.2. Two-in-one: determinization and state reduction

Let $\mathscr{A} = (A, \delta, \sigma, \tau)$ be a fuzzy automaton and $\varphi$ a fuzzy relation on $A$. For each $u \in X^*$ we define a fuzzy set $\varphi_u : A \to L$ inductively, as follows: for the empty word $\varepsilon$ and all $u \in X^*$ and $x \in X$ we set

$$\varphi_\varepsilon = \sigma \circ \varphi, \qquad \varphi_{ux} = \varphi_u \circ \delta_x \circ \varphi \tag{5.11}$$

Clearly, if $u = x_1 \dots x_n$, where $x_1, \dots, x_n \in X$, then

$$\varphi_u = \sigma \circ \varphi \circ \delta_{x_1} \circ \varphi \circ \dots \circ \delta_{x_n} \circ \varphi. \tag{5.12}$$

Now, set $A_\varphi = \{\varphi_u \mid u \in X^*\}$, and define $\delta_\varphi : A_\varphi \times X \to A_\varphi$ and $\tau_\varphi : A_\varphi \to L$ as follows:

$$\delta_\varphi(\varphi_u, x) = \varphi_{ux}, \qquad \tau_\varphi(\varphi_u) = \varphi_u \circ \tau, \tag{5.13}$$

for all $u \in X^*$ and $x \in X$. If $\varphi_u = \varphi_v$, for some $u, v \in X^*$, then for each $x \in X$ we have that

$$\delta_\varphi(\varphi_u, x) = \varphi_{ux} = \varphi_u \circ \delta_x \circ \varphi = \varphi_v \circ \delta_x \circ \varphi = \varphi_{vx} = \delta_\varphi(\varphi_v, x),$$

and hence, $\delta_\varphi$ is a well-defined function. It is clear that $\tau_\varphi$ is also a well-defined function, and consequently, $\mathscr{A}_\varphi = (A_\varphi, \delta_\varphi, \varphi_\varepsilon, \tau_\varphi)$ is a well-defined crisp-deterministic fuzzy automaton.

The main question that arises here is how to choose a fuzzy relation $\varphi$ so that the automaton $\mathscr{A}_\varphi$ is equivalent to the original automaton $\mathscr{A}$. The following theorem gives an answer to this question.

**Theorem 5.1.** *Let $\mathscr{A} = (A, \delta, \sigma, \tau)$ be a fuzzy automaton and $\varphi$ a reflexive weakly right invariant fuzzy relation on $\mathscr{A}$. Then $\mathscr{A}_\varphi = (A_\varphi, \delta_\varphi, \varphi_\varepsilon, \tau_\varphi)$ is an accessible crisp-deterministic fuzzy automaton equivalent to $\mathscr{A}$.*

*Proof.* According to (5.7), by induction we easily prove that

$$\tau_{x_1 x_2 \ldots x_n} = \varphi \circ \delta_{x_1} \circ \varphi \circ \cdots \circ \delta_{x_n} \circ \varphi \circ \tau. \tag{5.14}$$

for each $n \in \mathbb{N}$ and all $x_1, \ldots, x_n \in X$.

Now, according to (5.14), for each $u = x_1 \ldots x_n$, where $x_1, \ldots, x_n \in X$, we have that

$$\begin{aligned}
[\![\mathscr{A}_\varphi]\!](u) = \tau_\varphi(\delta_\varphi(\varphi_\varepsilon, u)) = \tau_\varphi(\varphi_u) = \varphi_u \circ \tau = \\
= (\sigma \circ \varphi \circ \delta_{x_1} \circ \varphi \circ \ldots \circ \delta_{x_n} \circ \varphi) \circ \tau = \\
= \sigma \circ (\varphi \circ \delta_{x_1} \circ \varphi \circ \ldots \circ \delta_{x_n} \circ \varphi \circ \tau) = \\
= \sigma \circ \tau_u = \sigma \circ \delta_u \circ \tau = [\![\mathscr{A}]\!](u),
\end{aligned}$$

and besides,

$$[\![\mathscr{A}_\varphi]\!](\varepsilon) = \tau_\varphi(\varphi_\varepsilon) = \varphi_\varepsilon \circ \tau = \sigma \circ \varphi \circ \tau = \sigma \circ \tau = [\![\mathscr{A}]\!](\varepsilon),$$

which means that $[\![\mathscr{A}_\varphi]\!] = [\![\mathscr{A}]\!]$. Therefore, $\mathscr{A}_\varphi$ is equivalent to $\mathscr{A}$. □

In addition, the following is true.

**Theorem 5.2.** *Let $\mathscr{A} = (A, \delta^A, \sigma^A, \tau^A)$ be a fuzzy automaton and $\varphi$ a weakly right invariant fuzzy quasi-order on $\mathscr{A}$. Then the automaton $\mathscr{A}_\varphi$ is isomorphic to the Nerode automaton of the afterset fuzzy automaton $\mathscr{A}/\varphi$.*

*Proof.* For the sake of simplicity, set $B = A/\varphi$ and $\mathscr{B} = \mathscr{A}/\varphi$, i.e., let $\mathscr{B} = (B, \delta^B, \sigma^B, \tau^B)$ be the afterset fuzzy automaton of $\mathscr{A}$ corresponding to $\varphi$. Consider the Nerode automaton $\mathscr{B}_N = (B_N, \delta_N, \sigma^B_\varepsilon, \tau_N)$ of $\mathscr{B}$.

First, by induction on the length of a word, we will prove that for any $u \in X^*$ the following is true:

$$\sigma^B_u(a\varphi) = \varphi_u(a), \quad \text{for every } a \in A. \tag{5.15}$$

For any $a \in A$ we have that $\sigma^B_\varepsilon(a\varphi) = \sigma^B(a\varphi) = (\sigma^A \circ \varphi)(a) = \varphi_\varepsilon(a)$, and thus, (5.15) holds when $u$ is the empty word. Next, suppose that (5.15) holds for some word $u \in X^*$. By (5.12) and idempotency of $\varphi$ it follows that $\varphi_u \circ \varphi = \varphi_u$, so for each $x \in X$ and each $a \in A$ we have that

$$\begin{aligned}
\sigma^B_{ux}(a\varphi) = (\sigma^B_{ux} \circ \delta^B_x)(a\varphi) = \bigvee_{b \in A} \sigma^B_u(b\varphi) \otimes \delta^B_x(b\varphi, a\varphi) = \\
= \bigvee_{b \in A} \varphi_u(b) \otimes (\varphi \circ \delta^A_x \circ \varphi)(b, a) = (\varphi_u \circ \varphi \circ \delta^A_x \circ \varphi)(a) = \\
= (\varphi_u \circ \delta^A_x \circ \varphi)(a) = \varphi_{ux}(a).
\end{aligned}$$

Therefore, we conclude that (5.15) holds for every $u \in X^*$.

Now, define a function $\xi : A_\varphi \to B_N$ by $\xi(\varphi_u) = \sigma^B_u$, for each $u \in X^*$. For arbitrary $u, v \in X^*$ we have that

$$\varphi_u = \varphi_v \;\Leftrightarrow\; (\forall a \in A)\ \varphi_u(a) = \varphi_v(a) \;\Leftrightarrow\; (\forall a \in A)\ \sigma^B_u(a\varphi) = \sigma^B_v(a\varphi) \;\Leftrightarrow\; \sigma^B_u = \sigma^B_v,$$

so $\xi$ is a well-defined and injective function. It is clear that $\xi$ is also surjective, and therefore, $\xi$ is a bijective function. Also, for all for all $u \in X^*$ and $x \in X$ we have that

$$\begin{aligned}
&\delta_N(\xi(\varphi_u), x) = \delta_N(\sigma^B_u, x) = \sigma^B_{ux} = \xi(\varphi_{ux}) = \xi(\delta_\varphi(\varphi_u, x)),\\
&\tau_N(\xi(\varphi_u)) = \tau_N(\sigma^B_u) = \sigma^B_u \circ \tau^B = [\![\mathscr{B}]\!](u) =\\
&\qquad\qquad = [\![\mathscr{A}]\!](u) = [\![\mathscr{A}_\varphi]\!](u) = \varphi_u \circ \tau^A = \tau_\varphi(\varphi_u),
\end{aligned}$$

so $\xi$ is an isomorphism of the automaton $\mathscr{A}_\varphi$ onto the Nerode automaton $\mathscr{B}_N$ of $\mathscr{B} = \mathscr{A}/\varphi$. □

*Remark 5.1.* Note that in the previous theorem we need $\varphi$ to be weakly right invariant only to prove that $\tau_N(\xi(\varphi_u)) = \tau_\varphi(\varphi_u)$. Everything else can be proved under the weaker assumpton that $\varphi$ is a fuzzy quasi-order.

In the case when working with right invariant fuzzy quasi-orders, it is possible to compare the size of the corresponding automata. This follows from the following theorem.

**Theorem 5.3.** *Let $\mathscr{A} = (A, \delta, \sigma, \tau)$ be a fuzzy automaton and let $\varphi$ and $\phi$ be right invariant fuzzy quasi-orders on $\mathscr{A}$ such that $\varphi \leqslant \phi$. Then the automaton $\mathscr{A}_\phi$ is a homomorphic image of the automaton $\mathscr{A}_\varphi$.*

*Consequently, $|\mathscr{A}_\phi| \leqslant |\mathscr{A}_\varphi|$.*

*Proof.* First we note that $\varphi \leqslant \phi$ is equivalent to $\varphi \circ \phi = \phi \circ \varphi = \phi$, because $\varphi$ and $\phi$ are fuzzy quasi-orders.

Define a function $\xi : A_\varphi \to A_\phi$ by $\xi(\varphi_u) = \phi_u$, for each $u \in X^*$. First we prove that $\xi$ is well-defined. Let $u, v \in X^*$ such that $\varphi_u = \varphi_v$. According to (5.9) and (5.12), by induction we easily prove that $\varphi_w = \sigma_w \circ \varphi$ and $\phi_w = \sigma_w \circ \phi$, for every $w \in X^*$, whence

$$\phi_u = \sigma_u \circ \phi = \sigma_u \circ \varphi \circ \phi = \varphi_u \circ \phi = \varphi_v \circ \phi = \sigma_v \circ \varphi \circ \phi = \sigma_v \circ \phi = \phi_v.$$

Therefore, $\xi$ is a well-defined function. It is clear that $\xi$ is a surjective function. Moreover, it is evident that $\delta_\phi(\xi(\varphi_u), x) = \xi(\delta_\varphi(\varphi_u, x))$, for all $u \in X^*$ and $x \in X$, and

$$\begin{aligned}
\tau_\phi(\xi(\varphi_u)) = \tau_\phi(\phi_u) = \phi_u \circ \tau = [\![\mathscr{A}_\phi]\!](u) =\\
= [\![\mathscr{A}]\!](u) = [\![\mathscr{A}_\varphi]\!](u) = \varphi_u \circ \tau = \tau_\varphi(\varphi_u),
\end{aligned}$$

and hence, $\xi$ is a homomorphism of $\mathscr{A}_\varphi$ onto $\mathscr{A}_\phi$ and $|\mathscr{A}_\phi| \leqslant |\mathscr{A}_\varphi|$. □

Note that when $\varphi$ is a reflexive weakly left invariant fuzzy relation on a fuzzy automaton $\mathscr{A}$, then $\mathscr{A}_\varphi$ is just the Nerode automaton of $\mathscr{A}$, and we do not get any new construction. Besides, the following is true.

**Theorem 5.4.** *Let $\mathscr{A} = (A, \delta^A, \sigma^A, \tau^A)$ be a fuzzy automaton and $\varphi$ a weakly left invariant fuzzy quasi-order on $\mathscr{A}$. Then the Nerode automaton of the afterset fuzzy automaton $\mathscr{A}/\varphi$ is isomorphic to the Nerode automaton of $\mathscr{A}$.*

*Proof.* For the sake of simplicity, set $B = A/\varphi$ and $\mathscr{B} = \mathscr{A}/\varphi$, i.e., let $\mathscr{B} = (B, \delta^B, \sigma^B, \tau^B)$ be the afterset fuzzy automaton of $\mathscr{A}$ corresponding to $\varphi$.

First, by induction on the length of a word, we will prove that for any $u \in X^*$ the following is true:

$$\sigma^B_u(a\varphi) = \sigma^A_u(a), \quad \text{for every } a \in A. \tag{5.16}$$

For every $a \in A$ we have that $\sigma^B_\varepsilon(a\varphi) = (\sigma^A \circ \varphi)(a) = \sigma^A(a) = \sigma^A_\varepsilon(a)$, so (5.16) holds when $u$ is the empty word. Next, suppose that (5.16) holds for some word $u \in X^*$. According to (5.16) and our starting hypothesis that $\varphi$ is a weakly left invariant fuzzy quasi-order, for all $x \in X$ and $a \in A$ we obtain that

$$\begin{aligned}\sigma^B_{ux}(a\varphi) = (\sigma^B_u \circ \delta^B_x)(a\varphi) = \bigvee_{b\in A} \sigma^B_u(b\varphi) \otimes \delta^B_x(b\varphi, a\varphi) = \bigvee_{b\in A} \sigma^A_u(b) \otimes (\varphi \circ \delta^A_x \circ \varphi)(b,a)\\ = (\sigma^A_u \circ \varphi \circ \delta^A_x \circ \varphi)(a) = (\sigma^A_u \circ \delta^A_x \circ \varphi)(a) = (\sigma^A_{ux} \circ \varphi)(a) = \sigma^A_{ux}(a),\end{aligned}$$

what completes the proof of (5.16).

Now, define a function $\xi : B_N \to A_N$ by $\xi(\sigma^B_u) = \sigma^A_u$, for each $u \in X^*$. For arbitrary $u, v \in X^*$ we have that

$$\sigma^B_u = \sigma^B_v \;\Leftrightarrow\; (\forall a \in A)\, \sigma^B_u(a\varphi) = \sigma^B_v(a\varphi) \;\Leftrightarrow\; (\forall a \in A)\, \sigma^A_u(a) = \sigma^A_v(a) \;\Leftrightarrow\; \sigma^A_u = \sigma^a_v,$$

which means that $\xi$ is a well-defined and injective function. In addition, $\xi$ is surjective, and consequently, $\xi$ is a bijective function. It is easy to check that $\xi$ is a homomorphism, and therefore, $\xi$ is an isomorphism of the Nerode automaton of $\mathscr{B}$ onto the Nerode automaton of $\mathscr{A}$. □

As we said earlier, when $\varphi$ is a reflexive weakly left invariant fuzzy relation on a fuzzy automaton $\mathscr{A}$, then $\mathscr{A}_\varphi$ is just the Nerode automaton of $\mathscr{A}$, and we do not get any new construction. However, we will show that weakly left invariant fuzzy relations work well with another construction, and can be very useful in the determinization of the reverse fuzzy automaton of $\mathscr{A}$.

Let $\mathscr{A} = (A, \delta, \sigma, \tau)$ be a fuzzy automaton and $\psi$ a fuzzy relation on $A$. For each $u \in X^*$ we define a fuzzy set $\psi^u : A \to L$ inductively, as follows: for the empty word $\varepsilon$ and all $u \in X^*$ and $x \in X$ we set

$$\psi^\varepsilon = \psi \circ \tau, \qquad \psi^{xu} = \psi \circ \delta_x \circ \psi^u \tag{5.17}$$

Clearly, if $u = x_1 \dots x_n$, where $x_1, \dots, x_n \in X$, then

$$\psi^u = \psi \circ \delta_{x_1} \circ \psi \circ \ldots \circ \delta_{x_n} \circ \psi \circ \tau. \tag{5.18}$$

Now, set $A^\psi = \{\psi_u \mid u \in X^*\}$, and define $\delta^\psi : A^\psi \times X \to A^\psi$ and $\tau^\psi : A^\psi \to L$ as follows:

$$\delta^\psi(\psi^u, x) = \psi^{xu}, \qquad \tau^\psi(\psi^u) = \sigma \circ \psi^u, \tag{5.19}$$

for all $u \in X^*$ and $x \in X$. If $\psi_u = \psi_v$, for some $u, v \in X^*$, then for each $x \in X$ we have that

$$\delta^\psi(\psi^u, x) = \psi^{xu} = \psi \circ \delta_x \circ \psi^u = \psi \circ \delta_x \circ \psi^v = \psi^{xv} = \delta^\psi(\psi^v, x),$$

and hence, $\delta^\psi$ is a well-defined function. Clearly, $\tau^\psi$ is also a well-defined function, so $\mathscr{A}^\psi = (A^\psi, \psi^\varepsilon, \delta^\psi, \tau^\psi)$ is a well-defined crisp-deterministic fuzzy automaton.

Now we prove that the following is true.

**Theorem 5.5.** *Let $\mathscr{A} = (A, \delta, \sigma, \tau)$ be a fuzzy automaton and $\psi$ a reflexive weakly left invariant fuzzy relation on $\mathscr{A}$. Then $\mathscr{A}^\psi = (A^\psi, \delta^\psi, \psi^\varepsilon, \tau^\psi)$ is an accessible crisp-deterministic fuzzy automaton equivalent to the reverse fuzzy automaton of $\mathscr{A}$.*

*Proof.* Consider an arbitrary word $u = x_1 \ldots x_n$, where $x_1, \ldots, x_n \in X$. Using (5.18) we obtain that

$$\sigma \circ \psi \circ \delta_{x_n} \circ \psi \circ \cdots \circ \delta_{x_1} \circ \psi = \sigma_{x_n \ldots x_1},$$

whence it follows that

$$\begin{aligned}
[\![\mathscr{A}^\psi]\!](u) &= \tau^\psi(\delta^\psi(\psi^\varepsilon, u)) = \tau^\psi(\psi^{\bar{u}}) = \\
&= \sigma \circ \psi^{\bar{u}} = \sigma \circ (\psi \circ \delta_{x_n} \circ \psi \circ \ldots \circ \delta_{x_1} \circ \psi \circ \tau) = \\
&= (\sigma \circ \psi \circ \delta_{x_n} \circ \psi \circ \ldots \circ \delta_{x_1} \circ \psi) \circ \tau = \sigma_{x_n \ldots x_1} \circ \tau = \\
&= \sigma_{\bar{u}} \circ \tau = [\![\mathscr{A}]\!](\bar{u}) = [\![\overline{\mathscr{A}}]\!](u).
\end{aligned}$$

On the other hand,

$$[\![\mathscr{A}^\psi]\!](\varepsilon) = \tau^\psi(\psi^\varepsilon) = \sigma \circ \psi^\varepsilon = \sigma \circ \psi \circ \tau = \sigma \circ \tau = [\![\mathscr{A}]\!](\varepsilon) = [\![\overline{\mathscr{A}}]\!](\varepsilon).$$

Therefore, $[\![\mathscr{A}^\psi]\!] = [\![\overline{\mathscr{A}}]\!]$, i.e., $\mathscr{A}^\psi$ is equivalent to $\overline{\mathscr{A}}$. □

The next two theorems can be proved similarly as Theorems 5.2 and 5.3, so they proofs will be omitted.

**Theorem 5.6.** *Let $\mathscr{A} = (A, \delta, \sigma, \tau)$ be a fuzzy automaton and $\psi$ a weakly left invariant fuzzy quasi-order on $\mathscr{A}$. Then the automaton $\mathscr{A}^\psi$ is isomorphic to the reverse Nerode automaton of the afterset fuzzy automaton $\mathscr{A}/\psi$.*

**Theorem 5.7.** *Let $\mathscr{A} = (A, \delta, \sigma, \tau)$ be a fuzzy automaton and let $\psi$ and $\psi'$ be left invariant fuzzy quasi-orders on $\mathscr{A}$ such that $\psi \leqslant \psi'$. Then the automaton $\mathscr{A}^{\psi'}$ is a homomorphic image of the automaton $\mathscr{A}^\psi$.*

Note that the reverse Nerode automaton plays a crucial role in Brzozowski type determinization of a fuzzy automaton. Namely, it has been proven in [60] that when we start from a fuzzy automaton $\mathscr{A}$, two consecutive applications of the construction of a reverse Nerode automaton produce a minimal crisp-deterministic fuzzy automaton which is equivalent to $\mathscr{A}$. We will show here that the first of these two constructions can be replaced by construction of the automaton $\mathscr{A}^{\psi}$, for some reflexive weakly left invariant fuzzy relation $\psi$ on $\mathscr{A}$.

**Theorem 5.8.** *Let $\mathscr{A} = (A, \delta, \sigma, \tau)$ be a fuzzy automaton and $\psi$ a reflexive weakly left invariant fuzzy relation on $\mathscr{A}$. Then the reverse Nerode automaton of $\mathscr{A}^{\psi}$ is a minimal crisp-deterministic fuzzy automaton equivalent to $\mathscr{A}$.*

*Proof.* As we have proved in Theorem 5.5, $\mathscr{A}^{\psi}$ is an accessible crisp-deterministic fuzzy automaton equivalent to $\overline{\mathscr{A}}$. According to Theorem 3.5 [60], for any accessible crisp-deterministic fuzzy automaton $\mathscr{B}$, the reverse Nerode automaton of $\mathscr{B}$ is a minimal crisp-deterministic fuzzy automaton equivalent to $\overline{\mathscr{B}}$. Therefore, $\mathscr{A}^{\psi}$ is a minimal crisp-deterministic fuzzy automaton equivalent to the reverse of $\overline{\mathscr{A}}$, i.e., it is a minimal crisp-deterministic fuzzy automaton equivalent to $\mathscr{A}$. □

As the automaton $\mathscr{A}^{\psi}$ can be significantly smaller than the reverse Nerode automaton $\mathscr{A}_{\overline{N}}$ (in particular, if $\psi$ is the greatest left invariant fuzzy quasi-order on $\mathscr{A}$), replacing $\mathscr{A}_{\overline{N}}$ with $\mathscr{A}^{\psi}$ in the first step of Brzozowski type procedure we could mitigate a combinatorial blow up of the number of states that may happen in this step. In the second step, such a problem does not exist because both constructions give the minimal crisp-deterministic fuzzy automaton equivalent to $\mathscr{A}$.

Let $\mathscr{A} = (A, \delta, \sigma, \tau)$ be a fuzzy automaton over an alphabet $X = \{x_1, \ldots, x_m\}$ and $\varphi$ a fuzzy relation on $A$. For each $u \in X^*$ define an $(m+1)$-tuple $\varphi^c_u$ by

$$\varphi^c_u = (\varphi_{ux_1}, \ldots, \varphi_{ux_m}, \varphi_u \circ \tau) = (\varphi_u \circ \delta_{x_1}, \ldots, \varphi_u \circ \delta_{x_m}, \varphi_u \circ \tau),$$

set $A^c_\varphi = \{\varphi^c_u \mid u \in X^*\}$, and define $\delta^c_\varphi : A^c_\varphi \times X \to A^c_\varphi$ and $\tau^c_\varphi : A^c_\varphi \to L$ as follows:

$$\delta^c_\varphi(\varphi^c_u, x) = \varphi^c_{ux}, \qquad \tau^c_\varphi(\varphi^c_u) = \varphi_u \circ \tau, \tag{5.20}$$

for all $u \in X^*$ and $x \in X$. We have the following:

**Theorem 5.9.** *Let $\mathscr{A} = (A, \delta, \sigma, \tau)$ be a fuzzy automaton over an alphabet $X = \{x_1, \ldots, x_m\}$ and let $\varphi$ be a weakly right invariant fuzzy relation on $A$. Then $\mathscr{A}^c_\varphi = (A^c_\varphi, \delta^c_\varphi, \varphi^c_\varepsilon, \tau^c_\varphi)$ is an acessible crisp-deterministic fuzzy automaton equivalent to $\mathscr{A}$.*

*Proof.* Let $\varphi^c_u = \varphi^c_v$, for some $u, v \in X^*$. This means that $\varphi_{ux_i} = \varphi_{vx_i}$, for any $i \in \{1, \ldots, m\}$, and $\varphi_u \circ \tau = \varphi_v \circ \tau$. Now, for an arbitrary $x \in X$ we have that $\varphi_{ux} = \varphi_{vx}$, whence

$$\varphi_{uxx_i} = \varphi_{ux} \circ \delta \circ \varphi = \varphi_{vx} \circ \delta_{x_i} \circ \varphi = \varphi_{vxx_i},$$

for each $i \in \{1,\ldots,m\}$, and also, $\varphi_{ux} \circ \tau = \varphi_{vx} \circ \tau$. Hence, $\varphi^c_{ux} = \varphi^c_{vx}$, so $\delta^c_\varphi(\varphi^c_u, x) = \varphi^c_{ux} = \varphi^c_{vx} = \delta^c_\varphi(\varphi^c_v, x)$, and this means that $\delta^c_\varphi$ is a well-defined function. Clearly, $\tau^c_\varphi$ is also a well-defined function, and consequently, $\mathscr{A}^c_\varphi$ is an accessible crisp-deterministic automaton.

Next, for each $u \in X^*$ we have that

$$[\![\mathscr{A}^c_\varphi]\!](u) = \tau^c_\varphi(\delta^c_\varphi(\varphi^c_\varepsilon, u)) = \tau^c_\varphi(\varphi^c_u) = \varphi_u \circ \tau = [\![\mathscr{A}_\varphi]\!](u) = [\![\mathscr{A}]\!](u),$$

and therefore, $[\![\mathscr{A}^c_\varphi]\!]$ is equivalent to $\mathscr{A}$. □

In the case when $\varphi$ is the crisp equality on $A$, we have that $\varphi_u = \sigma^A_u$, for each $u \in X^*$, and $\mathscr{A}^c_\varphi$ is the automaton constructed in Chapter 3, where it was called the *reduced Nerode automaton* of $\mathscr{A}$. Here we will use a different terminology. Namely, the first $m$ elements in the $(m+1)$-tuple $\varphi^c_u$ are the children of the vertex $\varphi_u$ in the transition tree of the automaton $\mathscr{A}_\varphi$ (see Algorithm 5.15), and the $m+1$st element of $\varphi^c_u$ is the termination degree of $\varphi_u$ in $\mathscr{A}_\varphi$. For this reason, the automaton $\mathscr{A}^c_\varphi$ will be called the *children automaton* of $\mathscr{A}_\varphi$. In this regard, the above mentioned automaton constructed in Chapter 3 is the children automaton of the Nerode automaton $\mathscr{A}_N$ of $\mathscr{A}$. In the sequel, the children automaton of $\mathscr{A}_N$ will be denoted by $\mathscr{A}^c_N = (A^c_N, \delta^c_N, \sigma^c_\varepsilon, \tau^c_N)$.

It is important to note that in the case of Nerode and reduced Nerode automaton we proved that the number of states of Nerode automaton is always greater or equal to the number of states of the corresponding reduced Nerode automaton, and that Nerode automaton is finite if and only the corresponding reduced Nerode automaton is finite. Here we will prove that analogous results holds true for automaton $\mathscr{A}_\varphi$ and children automaton $\mathscr{A}^c_\varphi$ of $\mathscr{A}_\varphi$.

**Theorem 5.10.** *Let $\mathscr{A} = (A, \delta, \sigma, \tau)$ be a fuzzy automaton over an alphabet $X = \{x_1,\ldots,x_m\}$ and $\varphi$ a fuzzy relation on $A$. Then $|\mathscr{A}^c_\varphi| \leqslant |\mathscr{A}_\varphi|$.*

*Proof.* Let us define a mapping $\phi : A_\varphi \to A^c_\varphi$ by

$$\phi(\varphi_u) = \varphi^c_u,$$

for every $u \in X^*$. Let $u, v \in X^*$ such that $\varphi_u = \varphi_v$. It is easy to prove that for every $x \in X$, $\varphi_{ux} = \varphi_{vx}$ and hence

$$\phi(\varphi_u) = \varphi^c_u = (\varphi_{ux_1},\ldots,\varphi_{ux_m}, \varphi_u \circ \tau) = (\varphi_{vx_1},\ldots,\varphi_{vx_m}, \varphi_v \circ \tau) = \varphi^c_v = \phi(\varphi_v),$$

so we have that $\phi$ is well-defined mapping. It is clear that $\phi$ is surjective, and accordingly, $|\mathscr{A}^c_\varphi| \leqslant |\mathscr{A}_\varphi|$. □

**Theorem 5.11.** *Let $\mathscr{A} = (A, \delta, \sigma, \tau)$ be a fuzzy automaton over an alphabet $X = \{x_1,\ldots,x_m\}$ and $\varphi$ a fuzzy relation on $A$. Then automaton $\mathscr{A}^c_\varphi$ is finite if and only if automaton $\mathscr{A}_\varphi$ is finite.*

*Proof.* Let $A^c_\varphi = \{\varphi^c_{u_1}, \ldots, \varphi^c_{u_k}\}$, for some $k \in \mathbb{N}$ and $u_1, \ldots, u_k \in X^*$. For every $u \in X^+$ we have that $u = vx$, for some $v \in X^*$ and $x \in X$, and $\varphi^c_v = \varphi^c_{u_j}$, for some $j \in \{1, \ldots, k\}$, so $\varphi_u = \varphi_{vx} = \varphi^x_{u_j}$. Hence, $A^c_\varphi \setminus \{\varphi_\varepsilon\} \subseteq \{\varphi_{u_j x_l} \mid 1 \leqslant j \leqslant k, 1 \leqslant l \leqslant m\}$, and therefore, $|A_\varphi| \leqslant km + 1$, i.e., $|\mathscr{A}_\varphi| \leqslant |\mathscr{A}^c_\varphi| \cdot |X| + 1$.

Proof of the fact that $\mathscr{A}^c_\varphi$ is finite whenever $\mathscr{A}_\varphi$ is finite follows immediately by Theorem 5.10. □

**Theorem 5.12.** *Let $\mathscr{A} = (A, \delta, \sigma, \tau)$ be a fuzzy automaton over an alphabet $X = \{x_1, \ldots, x_m\}$ and let $\varphi$ and $\phi$ be right invariant fuzzy quasi-orders on $A$ such that $\varphi \leqslant \phi$. Then the automaton $\mathscr{A}^c_\phi$ is a homomorphic image of the automaton $\mathscr{A}^c_\varphi$, and consequently, $|\mathscr{A}^c_\phi| \leqslant |\mathscr{A}^c_\varphi|$.*

*Proof.* Define a function $\xi : A^c_\varphi \to A^c_\phi$ by $\xi(\varphi^c_u) = \phi^c_u$, for each $u \in X^*$. Let $\varphi^c_u = \varphi^c_v$, for some $u, v \in X^*$, i.e., let $\varphi_{ux} = \varphi_{vx}$, for eny $x \in X$, and $\varphi_u \circ \tau = \varphi_v \circ \tau$. As in the proof of Theorem 5.3 we obtain that $\varphi_w = \sigma_w \circ \varphi$ and $\phi_w = \sigma_w \circ \phi$, for every $w \in X^*$, and for any $x \in X$ we have that

$$\phi_{ux} = \sigma_{ux} \circ \phi = \sigma_{ux} \circ \varphi \circ \phi = \varphi_{ux} \circ \phi = \varphi_{vx} \circ \phi = \sigma_{vx} \circ \varphi \circ \phi = \sigma_{vx} \circ \phi = \phi_{vx},$$

and also,

$$\phi_u \circ \tau = [\![\mathscr{A}_\phi]\!](u) = [\![\mathscr{A}]\!](u) = [\![\mathscr{A}_\varphi]\!](u) = \varphi_u \circ \tau = \\ = \varphi_v \circ \tau = [\![\mathscr{A}_\varphi]\!](v) = [\![\mathscr{A}_\phi]\!](v) = \phi_v \circ \tau.$$

Therefore, $\phi^c_u = \phi^c_v$, which means that $\xi$ is a well-defined function, and clearly, $\xi$ is surjective. Moreover, for all $u \in X^*$ and $x \in X$ we have that

$$\xi(\delta^c_\varphi(\varphi^c_u, x)) = \xi(\varphi^c_{ux}) = \phi^c_{ux} = \delta^c_\phi(\phi^c_u, x) = \delta^c_\phi(\xi(\varphi^c_u), x),$$

and, on the other hand,

$$\tau^c_\phi(\phi^c_u) = \phi_u \circ \tau = [\![\mathscr{A}_\phi]\!](u) = [\![\mathscr{A}]\!](u) = [\![\mathscr{A}_\varphi]\!](u) = \varphi_u \circ \tau = \tau^c_\varphi(\varphi^c_u).$$

Hence, $\xi$ is a homomorphism of $\mathscr{A}_\varphi$ onto $\mathscr{A}_\phi$. □

**Theorem 5.13.** *Let $\mathscr{A} = (A, \delta^A, \sigma^A, \tau^A)$ be a fuzzy automaton over an alphabet $X = \{x_1, \ldots, x_m\}$ and let $\varphi$ be a weakly right invariant fuzzy quasi-order on $\mathscr{A}$. Then the automaton $\mathscr{A}^c_\varphi$ is isomorphic to the automaton $\mathscr{B}^c_N$, where $\mathscr{B} = \mathscr{A}/\varphi$ is the afterset fuzzy automaton of $\mathscr{A}$ with respect to $\varphi$.*

*Proof.* Define a function $\xi : A^c_\varphi \to B^c_N$ by $\xi(\varphi^c_u) = \sigma^{B,c}_u = (\sigma^B_{ux_1}, \ldots, \sigma^B_{ux_m}, \sigma^B_u \circ \tau^B)$, for each $u \in X^*$. As in the proof of Theorem 5.2 we obtain that $\sigma^B_u(a\varphi) = \varphi_u(a)$, for all $u \in X^*$ and $a \in A$, and by this it follows that

$$\begin{aligned} \varphi_{ux_i} = \varphi_{vx_i} \quad &\Leftrightarrow \quad (\forall a \in A)\ \varphi_{ux_i}(a) = \varphi_{vx_i}(a) \\ &\Leftrightarrow \quad (\forall a \in A)\ \sigma^B_{ux_i}(a\varphi) = \sigma^B_{vx_i}(a\varphi) \quad \Leftrightarrow \quad \sigma^B_{ux_i} = \sigma^B_{vx_i}, \end{aligned}$$

and also,

$$\begin{aligned}\varphi_u \circ \tau^A = \varphi_v \circ \tau^A \quad &\Leftrightarrow \quad [\![\mathscr{A}_\varphi]\!](u) = [\![\mathscr{A}_\varphi]\!](v) \quad \Leftrightarrow \quad [\![\mathscr{A}]\!](u) = [\![\mathscr{A}]\!](v) \\ &\Leftrightarrow \quad [\![\mathscr{B}]\!](u) = [\![\mathscr{B}]\!](v) \quad \Leftrightarrow \quad [\![\mathscr{B}_N]\!](u) = [\![\mathscr{B}_N]\!](v) \\ &\Leftrightarrow \quad \sigma_u^B \circ \tau^B = \sigma_v^B \circ \tau^B,\end{aligned}$$

for all $u, v \in X^*$ and $x_i \in X$, and therefore, $\varphi_u^c = \varphi_v^c$ if and only if $\sigma_u^{B,c} = \sigma_v^{B,c}$. This means that $\xi$ is a well-defined and injective function. It is clear that $\xi$ is also surjective, and iot can be easily verified that it is a homomorphism. Hence, $\xi$ is an isomorphism of $\mathscr{A}_\varphi^c$ onto $\mathscr{B}_N^c$. □

**Theorem 5.14.** *Let $\mathscr{A} = (A, \delta, \sigma, \tau)$ be a fuzzy automaton over an alphabet $X = \{x_1, \ldots, x_m\}$ and let $\varphi$ be a weakly right invariant fuzzy quasi-order on $A$. Then the automaton $\mathscr{A}_\varphi^c$ is a homomorphic image both of $\mathscr{A}_\varphi$ and $\mathscr{A}_N^c$, and consequently, $|\mathscr{A}_\varphi^c| \leqslant |\mathscr{A}_\varphi|$ and $|\mathscr{A}_\varphi^c| \leqslant |\mathscr{A}_N^c|$.*

*Proof.* According to Theorems 5.2 and 5.13, $\mathscr{A}_\varphi$ is isomorphic to $\mathscr{B}_N$, and $\mathscr{A}_\varphi^c$ is isomorphic to $\mathscr{B}_N^c$, and by Theorem 3.4 [61], $\mathscr{B}_N^c$ is a homomorphic image of $\mathscr{B}_N$. Therefore, $\mathscr{A}_\varphi^c$ is a homomorphic image of $\mathscr{A}_\varphi$.

On the other hand, $\mathscr{A}_N^c$ is isomorphic to $\mathscr{A}_\Delta^c$, where $\Delta$ is the crisp equality on $A$, and by Theorem 5.12, $\mathscr{A}_\varphi^c$ is a homomorphic image of $\mathscr{A}_N^c$. □

## 5.3. Two-in-one algorithms

Let $c_\vee$, $c_\wedge$, $c_\otimes$ and $c_\rightarrow$ be respectively computation times of the operations $\vee$, $\wedge$, $\otimes$ and $\rightarrow$ in $\mathscr{L}$. In particular, if $\mathscr{L}$ is linearly ordered, we can assume that $c_\vee = c_\wedge = 1$, and when $\mathscr{L}$ is the Gödel structure, we can also assume that $c_\otimes = c_\rightarrow = 1$.

**Algorithm 5.15** *[*Construction of the automaton $\mathscr{A}_\varphi$*]* The input of this algorithm are a fuzzy finite automaton $\mathscr{A} = (A, \delta, \sigma, \tau)$ with $n$ states, over a finite alphabet $X$ with $m$ letters, and a fuzzy relation $\varphi$ on $A$, and the output is the crisp-deterministic fuzzy automaton $\mathscr{A}_\varphi = (A_\varphi, \delta_\varphi, \varphi_\varepsilon, \tau_\varphi)$.

The procedure is to construct the *transition tree* of $\mathscr{A}_\varphi$ directly from $\mathscr{A}$, and during this procedure we use pointers $s(\cdot)$ which points vertices of the tree under construction to the corresponding integers. The transition tree of $\mathscr{A}_\varphi$ is constructed inductively as follows:

(A1) The root of the tree is $\varphi_\varepsilon = \sigma \circ \varphi$, and we put $T_0 = \{\varphi_\varepsilon\}$ and $s(\varphi_\varepsilon) = 1$, and we compute the value $\tau_\varphi(\varphi_\varepsilon) = \varphi_\varepsilon \circ \tau$.

(A2) After the $i$th step let a tree $T_i$ have been constructed, and vertices in $T_i$ have been labelled either 'closed' or 'non-closed'. The meaning of these two terms will be made clear in the sequel.

(A3) In the next step we construct a tree $T_{i+1}$ by enriching $T_i$ in the following way: for any non-closed leaf $\varphi_u$ occuring in $T_i$, where $u \in X^*$, and any $x \in X$ we add a vertex $\varphi_{ux} = \varphi_u \circ \delta_x \circ \varphi$ and an edge from $\varphi_u$ to $\varphi_{ux}$ labelled by $x$. Simultaneously, we check whether $\varphi_{ux}$ is a fuzzy set that has already been constructed. If it is true, if $\varphi_{ux}$ is equal to some previously computed $\varphi_v$, we mark $\varphi_{ux}$ as closed and set $s(\varphi_{ux}) = s(\varphi_v)$. Otherwise, we compute the value $\tau_\varphi(\varphi_{ux}) = \varphi_{ux} \circ \tau$ and set $s(\varphi_{ux})$ to be the next unassigned integer. The procedure terminates when all leaves are marked closed.

(A4) When the transition tree of $\mathscr{A}_\varphi$ is constructed, we erase all closure marks and glue leaves to interior vertices with the same pointer value. The diagram that results is the transition graph of $\mathscr{A}_\varphi$.

When $\varphi$ is taken to be the crisp equality on $A$, Algorithm 5.15 gives the Nerode automaton $\mathscr{A}_N$ of $\mathscr{A}$.

The above described procedure does necessarily terminates in a finite number of steps, since the collection $\{\varphi_u\}_{u\in X^*}$ may be infinite. However, in cases when this collection is finite, the procedure will terminate in a finite number of steps, after computing all its members. For instance, this holds if the subsemiring $\mathscr{L}^*(\delta,\sigma,\varphi)$ of $\mathscr{L}^*$ generated by all membership values taken by $\delta$, $\sigma$ and $\varphi$ is finite (but not only in this case). If $k$ denotes the number of elements of this subsemiring, then the collection $\{\varphi_u\}_{u\in X^*}$ can have at most $k^n$ different members.

The tree that is constructed by this algorithm is a full $m$-ary tree. At the end of the algorithm, the tree can contain at most $k^n$ internal vertices, and according to the well-known theorem on full $m$-ary trees, the total number of vertices is at most $mk^n + 1$. In the construction of any single vertice we can first perform a composition of the form $\varphi_u \circ \delta_x$, and then a composition of the form $(\varphi_u \circ \delta_x) \circ \varphi$, both of which have computation time $O(n^2(c_\otimes + c_\vee))$. Therefore, the computational time for all performed compositions is $O(mn^2k^n(c_\otimes + c_\vee))$. Moreover, the tree $T$ has at most $mk^n$ edges, and the computation time of their forming is $O(mk^n)$.

Time-consuming part of the procedure is the check whether the just computed fuzzy set is a copy of some previously computed fuzzy set. After we have constructed the $j$th fuzzy set, for some $j \in \mathbb{N}$ such that $2 \leqslant j \leqslant mk^n+1$, we compare it with the previously constructed fuzzy sets which correspond to non-closed vertices, whose number is at most $\min\{j-1, k^n\}$. Therefore, the total number of performed checks does not exceed $1+2+\cdots+k^n+(m-1)k^n \cdot k^n = \frac{1}{2}k^n(k^n+1)+(m-1)k^{2n}$. As the computational time of any single check is $O(n)$, the computational time for all performed checks is $O(mnk^{2n})$. Summarizing all the above we conclude that the computational time of the whole algorithm is $O(mnk^{2n})$, the same as the computational time of the part in which for any newly-constructed fuzzy set we check whether it is a copy of some previously computed fuzzy set.

Note that the number $k$ is characteristic of the fuzzy finite automaton $\mathscr{A}$, and it is not a general characteristic of the semiring $\mathscr{L}^*$ and its finitely

generated subsemirings. However, if we consider fuzzy automata over a finite lattice, the we can assume that $k$ is the number of elements of this lattice. Moreover, if $\mathscr{L}$ is the Gödel structure, then the set of all membership values taken by the fuzzy relations $\{\delta_x\}_{x\in X}$ and $\varphi$, and the fuzzy set $\sigma$ is a subsemiring of $\mathscr{L}^*$, and the number of these values does not exceed $mn^2+n$, so we can use that number instead of $k$.

In a similar way we can provide the following algorithm which constructs the automaton $\mathscr{A}^\psi$, for some fuzzy relation $\psi$ on $A$, and analyze its computation time.

**Algorithm 5.16** *[*Construction of the automaton $\mathscr{A}^\psi$*]* The input of this algorithm is a fuzzy finite automaton $\mathscr{A}=(A,\delta,\sigma,\tau)$ with $n$ states, over a finite alphabet $X$ with $m$ letters, and a fuzzy relation $\psi$ on $A$, and the output is the crisp-deterministic fuzzy automaton $\mathscr{A}^\psi=(A^\psi,\delta^\psi,\psi^\varepsilon,\tau^\psi)$.

The procedure is to construct the *transition tree* of $\mathscr{A}^\psi$ directly from $\mathscr{A}$, and during this procedure we use pointers $s(\cdot)$ which points vertices of the tree under construction to the corresponding integers. The transition tree of of $\mathscr{A}^\psi$ is constructed inductively as follows:

(A1) The root of the tree is $\psi^\varepsilon=\psi\circ\tau$, and we put $T_0=\{\psi^\varepsilon\}$ and $s(\psi^\varepsilon)=1$, and we compute the value $\tau^\psi(\psi^\varepsilon)=\sigma\circ\psi^\varepsilon$.

(A2) After the $i$th step let a tree $T_i$ have been constructed, and vertices in $T_i$ have been labelled either 'closed' or 'non-closed'. The meaning of these two terms will be made clear in the sequel.

(A3) In the next step we construct a tree $T_{i+1}$ by enriching $T_i$ in the following way: for any non-closed leaf $\psi^u$ occuring in $T_i$, where $u\in X^*$, and each $x\in X$ we add a vertex $\psi^{xu}=\psi\circ\delta_x\circ\psi^u$ and an edge from $\psi^u$ to $\psi^{xu}$ labelled by $x$. Simultaneously, we check whether $\psi^{xu}$ is a fuzzy set that has already been constructed. If it is true, if $\psi^{xu}$ is equal to some previously computed $\psi^v$, we mark $\psi^{xu}$ as closed and set $s(\psi^{xu})=s(\psi^v)$. Otherwise, we compute the value $\tau^\psi(\psi^{xu})=\sigma\circ\psi^{xu}$ and set $s(\psi^{xu})$ to be the next unassigned integer. The procedure terminates when all leaves are marked closed.

(A4) When the transition tree of $\mathscr{A}^\psi$ is constructed, we erase all closure marks and glue leaves to interior vertices with the same pointer value. The diagram that results is the transition graph of $\mathscr{A}^\psi$.

When $\psi$ is the crisp equality on $A$, Algorithm 5.15 produces the reverse Nerode automaton $\mathscr{A}_{\overleftarrow{N}}$ of $\mathscr{A}$.

The conditions under which the above procedure terminates in a finite number of steps and its computation time can be analyzed analogously as in Algorithm 5.15. The only difference is that instead of the subsemiring $\mathscr{L}^*(\delta,\sigma,\varphi)$ here we consider the subsemiring $\mathscr{L}^*(\delta,\tau,\psi)$ of $\mathscr{L}^*$ generated by all membership values taken by $\delta$, $\tau$ and $\psi$.

As we have said earlier, algorithms for computing the greatest right and left invariant fuzzy quasi-orders and the greatest weakly right and left invariant fuzzy quasi-orders on a fuzzy finite automaton were provided in

[116]. Here we present these algorithms and we perform an analysis of their computational time.

**Algorithm 5.17 (Computation of the greatest right invariant fuzzy quasi-order)** The input of this algorithm is a fuzzy finite automaton $\mathscr{A} = (A, \delta, \sigma, \tau)$ with $n$ states, over a finite alphabet $X$ with $m$ letters. The algorithm computes the greatest right invariant fuzzy quasi-order $\varphi^{\mathrm{ri}}$ on $\mathscr{A}$.

The procedure constructs the sequence of fuzzy quasi-orders $\{\varphi_k\}_{k\in\mathbb{N}}$, in the following way:

(A1) In the first step we set $\varphi_1 = \tau/\tau$.
(A2) After the $k$th step let $\varphi_k$ be the fuzzy quasi-order that has been constructed.
(A3) In the next step we construct the fuzzy quasi-order $\varphi_{k+1}$ by means of the formula

$$\varphi_{k+1} = \varphi_k \wedge \Big[\bigwedge_{x\in X} (\delta_x \circ \varphi_k)/(\delta_x \circ \varphi_k)\Big]. \tag{5.21}$$

(A4) Simultaneously, we check whether $\varphi_{k+1} = \varphi_k$.
(A5) When we find the smallest number $s$ such that $\varphi_{s+1} = \varphi_s$, the procedure of constructing the sequence $\{\varphi_k\}_{k\in\mathbb{N}}$ terminates and $\varphi^{\mathrm{ri}} = \varphi_s$.

If the subalgebra $\mathscr{L}(\delta, \tau)$ of $\mathscr{L}$, generated by all membership values taken by $\delta$ and $\tau$, satisfies DCC, the algorithm terminates in a finite number of steps.

Consider the computational time of this algorithm. In (A1) we compute $\tau^A/\tau^A$, which can be done in time $O(n^2 c_{\to})$. In (A3) we first compute all compositions $\delta_x \circ \varphi_k$, and if these computations are performed according to the definition of composition od fuzzy relations, their computational time is $O(mn^3(c_{\otimes} + c_{\vee}))$. Then we compute $\varphi_{k+1}$ by means of (5.21), and the computational time of this part is $O(mn^3(c_{\to} + c_{\wedge}))$. Thus, the total computational time of (A3) is $O(mn^3(c_{\to} + c_{\wedge} + c_{\otimes} + c_{\vee}))$. In (A4), the computational time to check whether $\varphi_{k+1} = \varphi_k$ is $O(n^2)$.

The hardest problem is to estimate the number of steps, in the case when it is finite. Consider fuzzy relations $\varphi_k$ as fuzzy matrices. After each step in the construction of the sequence $\{\varphi_k\}_{k\in\mathbb{N}}$ we check whether some entry has changed its value, and the algorithm terminates after the first step in which there was no change. Suppose that $\mathscr{L}(\delta, \tau)$ satisfies DCC. Then $\{\{\varphi_k(a,b)\}_{k\in\mathbb{N}} \mid (a,b) \in A^2\}$ is a finite collection of finite sequences, so there exists $s \in \mathbb{N}$ such that the number of different elements in each of these sequences is less than or equal to $s$. As the sequence $\{\varphi_k\}_{k\in\mathbb{N}}$ is descending, each entry can change its value at most $s-1$ times, and the total number of changes is less than or equal to $(s-1)(n^2-n)$ (the diagonal values must always be 1). Therefore, the algorithm terminates after at most $(s-1)(n^2-n)+2$ steps (in the first and last step values do not change).

Summing up, we get that the total computation time for the whole algorithm is $O(smn^5(c_{\to} + c_{\wedge} + c_{\otimes} + c_{\vee}))$, and hence, the algorithm is polynomial-time.

Let us note that the number $s$ is characteristic of the sequence $\{\varphi_k\}_{k\in\mathbb{N}}$, and in general it is not characteristic of the algebra $\mathscr{L}(\delta,\tau)$. However, in some cases the number of different elements in all descending chains in $\mathscr{L}(\delta,\tau)$ may have an upper bound $s$. For example, if the algebra $\mathscr{L}(\delta,\tau)$ is finite, then we can assume that $s$ is the number of elements of this algebra. In particular, if $\mathscr{L}$ is the Gödel structure, then the only values that can be taken by fuzzy relations $\{\varphi_k\}_{k\in\mathbb{N}}$ are 1 and those taken by $\delta$ and $\tau$. In this case, if $j$ is the number of all values taken by $\delta$ and $\tau$, then the algorithm terminates after at most $j(n^2-n)+2$ steps, and total computation time is $O(jmn^5)$. Since $j \leqslant mn^2+n^2$, total computation time can also be roughly expressed as $O(m^2n^7)$.

**Algorithm 5.18** ***(Computation of the greatest left invariant fuzzy quasi-order)*** The input of this algorithm is a fuzzy finite automaton $\mathscr{A}=(A,\delta,\sigma,\tau)$ with $n$ states, over a finite alphabet $X$ with $m$ letters. The algorithm computes the greatest left invariant fuzzy quasi-order $\psi^{\mathrm{li}}$ on $\mathscr{A}$.

The procedure constructs the sequence of fuzzy quasi-orders $\{\psi_k\}_{k\in\mathbb{N}}$, in the following way:

(A1) In the first step we set $\psi_1=\sigma\backslash\sigma$.
(A2) After the $k$th step let $\varphi_k$ be the fuzzy quasi-order that has been constructed.
(A3) In the next step we construct the fuzzy quasi-order $\psi_{k+1}$ by means of the formula

$$\psi_{k+1}=\psi_k\wedge\Big[\bigwedge_{x\in X}(\psi_k\circ\delta_x)\backslash(\psi_k\circ\delta_x)\Big].$$

(A4) Simultaneously, we check whether $\psi_{k+1}=\psi_k$.
(A5) When we find the smallest number $s$ such that $\psi_{s+1}=\psi_s$, the procedure of constructing the sequence $\{\psi_k\}_{k\in\mathbb{N}}$ terminates and $\psi^{\mathrm{li}}=\psi_s$.

The conditions under which this procedure terminates in a finite number of steps and its computation time can be analyzed analogously as in Algorithm 5.17.

**Algorithm 5.19** ***(Computation of the greatest weakly right invariant fuzzy quasi-order)*** The input of this algorithm is a fuzzy finite automaton $\mathscr{A}=(A,\delta,\sigma,\tau)$ with $n$ states, over a finite alphabet $X$ with $m$ letters. The algorithm computes the greatest weakly right invariant fuzzy quasi-order $\varphi^{\mathrm{wri}}$ on $\mathscr{A}$.

The procedure consists of two parts:

(A1) First we compute all members of the family $\{\tau_u \mid u\in X^*\}$, by means of Algorithm 4.5.
(A2) Then we compute $\varphi^{\mathrm{wri}}$ by means of formula

$$\varphi^{\mathrm{wri}}=\bigwedge_{u\in X^*}\tau_u/\tau_u.$$

Clearly, this procedure terminates in a finite number of steps under the same conditions as Algorithm 5.16. Under these conditions the computation time of the part (A1) is $O(mnk^{2n})$, and since it dominates over the computation time of (A2), which is $O(k^n c_\wedge + n^2 c_\to)$, we conclude that the computation time of the whole algorithm is $O(mnk^{2n})$, the same as for Algorithms 5.15 and 5.16.

Analogous analysis can be performed for the following algorithm that computes the greatest weakly left invariant fuzzy quasi-order on a fuzzy automaton.

**Algorithm 5.20** ***(Computation of the greatest weakly right invariant fuzzy quasi-order)*** The input of this algorithm is a fuzzy finite automaton $\mathscr{A} = (A, \delta, \sigma, \tau)$ over a finite alphabet $X$. The algorithm computes the greatest weakly left invariant fuzzy quasi-order $\psi^{\mathrm{wli}}$ on $\mathscr{A}$.

The procedure consists of two parts:

(A1) First we compute all members of the family $\{\sigma_u \mid u \in X^*\}$, using Algorithm 5.15.

(A2) Then we compute $\psi^{\mathrm{wli}}$ by means of formula

$$\psi^{\mathrm{wli}} = \bigwedge_{u \in X^*} \sigma_u \backslash \sigma_u.$$

Finally we construct the children automaton $\mathscr{A}_\varphi^c$.

**Algorithm 5.21** *[*Construction of the children automaton $\mathscr{A}_\varphi^c$*]* The input of this algorithm is a fuzzy finite automaton $\mathscr{A} = (A, \delta, \sigma, \tau)$ with $n$ states, over a finite alphabet $X = \{x_1, \ldots, x_m\}$ with $m$ letters, and the output is the children automaton $\mathscr{A}_\varphi^c = (A_\varphi^c, \delta_\varphi^c, \varphi_\varepsilon^c, \tau_\varphi^c)$, where $\varphi$ is the greatest weakly right invariant fuzzy quasi-order or the greatest right invariant fuzzy quasi-order on $\mathscr{A}$.

The procedure is to construct simultaneously the *transition tree* of $\mathscr{A}_\varphi$ and the *transition graph* of $\mathscr{A}_\varphi^c$ directly from $\mathscr{A}$. Except the pointer $s(\cdot)$ used in Algorithm 5.15, we also use another pointer $t(\cdot)$ which points vertices of the transition graph under construction to the corresponding integers. The transition tree of $\mathscr{A}_\varphi$ and the transition graph of $\mathscr{A}_\varphi^c$ are constructed in the following way:

(A1) We compute $\varphi$ using one of Algorithms 5.19 and 5.17, and construct the transition tree $T$ of $\mathscr{A}_\varphi$ using Algorithm 5.15.

(A2) To each non-closed vertex $\varphi_u$ of the tree $T$ we assign a vertex $\varphi_u^c$ of a graph $G$ as follows: When all children $\varphi_{ux_1}, \ldots, \varphi_{ux_m}$ of $\varphi_u$ in the tree $T$ are formed, we form the vertex $\varphi_u^c = (\varphi_{ux_1}, \ldots, \varphi_{ux_m}, \varphi_u \circ \tau)$ in the graph $G$. Simultaneously, we check whether $\varphi_u^c$ is an $(m+1)$-tuple that has already been constructed. If it is true, if $\varphi_u^c$ is equal to some previously computed $\varphi_v^c$, we mark $\varphi_u^c$ as closed and set $t(\varphi_u^c) = t(\varphi_v^c)$. Otherwise, we put $\tau_\varphi^c(\varphi_u^c) = \varphi_u \circ \tau$ and set $t(\varphi_u^c)$ to be the next integer that has not been used as a value for $t(\cdot)$.

(A3) For each non-closed vertex $\varphi_u^c$ of the graph $G$ and each $x \in X$, if $\varphi_v$ is a non-closed vertex in $T$ such that $s(\varphi_{ux}) = s(\varphi_v)$, in the graph $G$ we add an edge from $\varphi_u^c$ to $\varphi_v^c$ labelled by $x$.

(A4) When the graph $G$ is constructed, we glue closed vertices to non-closed vertices with the same pointer value, and erase closure marks. The diagram that results is the transition graph of $\mathscr{A}_\varphi^c$.

According to Theorems 5.2 and 5.13, $\mathscr{A}_\varphi$ is isomorphic to $\mathscr{B}_N$, where $\mathscr{B} = \mathscr{A}/\varphi$ is the afterset fuzzy automaton of $\mathscr{A}$ with respect to $\varphi$, and $\mathscr{A}_\varphi^c$ is isomorphic to $\mathscr{B}_N^c$, and by Theorem 3.7 [61], $\mathscr{B}_N^c$ is finite if and only if $\mathscr{B}_N$ is finite. Therefore, $\mathscr{A}_\varphi^c$ is finite if and only if $\mathscr{A}_\varphi$ is finite.

As in the analysis of Algorithm 5.15, consider the case when the subalgebra $\mathscr{L}(\delta, \sigma, \tau)$ of $\mathscr{L}$ is finite and has $l$ elements. As we have already seen, the computation time of the part (A1) is $O(mnl^{2n})$.

When in (A2) we construct a vertex of the graph $G$, as an $(m+1)$-tuple, all its components have already been computed during construction of the tree $T$, and all that remains to do is to point to them (or to their addresses). Hence, the computation time of forming every single vertex of the graph $G$ is $O(m)$, and since there are at most $l^n$ vertices in $G$, the computation time of forming all vertices of $G$ is $O(ml^n)$. During construction of the tree $T$ we also found which vertices in $T$ are equal as fuzzy sets, and they received the same pointer values. Therefore, when we check equality of two $(m+1)$-tuples in $A_\varphi^c$ we only need to check the equality of pointer values assigned to their components, and the computation time of such checking is $O(m)$. As the total number of checks performed in (A2) does not exceed $1 + 2 + \cdots + (l^n - 1) = \frac{1}{2}(l^n - 1)l^n$, we have that computation time of all checks performed in (A2) is $O(ml^{2n})$. Finally, in (A3) we form at most $ml^n$ edges in the graph $G$, and the computation time of this part is $O(ml^n)$.

Therefore, the most expensive part of this algorithm is (A1), and the computation time of the whole algorithm is $O(mnl^{2n})$, the same as for (A1), i.e., the same as for Algorithm 4.7.

## 5.4. Computational examples

In this section we provide several illustrative computational examples.

*Example 5.1.* Let $\mathscr{A}$ be a Boolean automaton over the two-element alphabet $X = \{x, y\}$ given by the transition graph shown in Fig. 5.1.

The transition tree and the transition graph of the Nerode automaton $\mathscr{A}_N$ of $\mathscr{A}$, constructed by means of Algorithm 5.15, are presented in Fig. 5.2. We see that the Nerode automaton $\mathscr{A}_N$ has 7 states.

By means of Algorithms 5.17 and 5.19 we compute the gratest right invariant fuzzy quasi-order $\varphi^{\mathrm{ri}}$ and the gratest weakly right invariant fuzzy

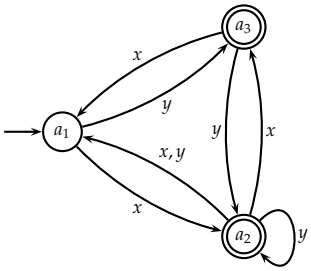



**Fig. 5.1** The transition graph of the fuzzy automaton $\mathscr{A}$ from Example 5.1.

quasi-order $\varphi^{\mathrm{wri}}$ on $\mathscr{A}$, which are represented by the following Boolan matrices:

$$\varphi^{\mathrm{ri}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}, \qquad \varphi^{\mathrm{wri}} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}.$$

Then, using Algorithm 5.15 we construct automata $\mathscr{A}_{\varphi^{\mathrm{ri}}}$ and $\mathscr{A}_{\varphi^{\mathrm{wri}}}$, whose transition trees and graphs are presented in Fig. 5.3 and 5.4, respectively. The automaton $\mathscr{A}_{\varphi^{\mathrm{ri}}}$ has 5 states, whereas the automaton $\mathscr{A}_{\varphi^{\mathrm{wri}}}$ has 3 states. Finally, using Algorithm 5.21 and the transition tree of $\mathscr{A}_N$ from Fig. 5.2, we construct the children automaton $\mathscr{A}_N^c$ of $\mathscr{A}_N$ presented in Fig. 5.4. Clearly, $\mathscr{A}_N^c$ is isomorphic to $\mathscr{A}_{\varphi^{\mathrm{ri}}}$.

According to Theorem 5.3, the number of states of the automaton $\mathscr{A}_{\varphi^{\mathrm{ri}}}$ is less than or equal to the number of states of the Nerode automaton $\mathscr{A}_N$, for every fuzzy automaton $\mathscr{A}$. This example shows that $\mathscr{A}_{\varphi^{\mathrm{ri}}}$ can be strictly smaller than $\mathscr{A}_N$. The example also shows that the greatest weakly right invariant fuzzy quasi-order $\varphi^{\mathrm{wri}}$ can give better results in determinization than the greatest weakly right invariant fuzzy quasi-order $\varphi^{\mathrm{ri}}$.

Finally, we find that all aftersets of fuzzy quasi-orders $\varphi^{\mathrm{ri}}$ and $\varphi^{\mathrm{wri}}$ are different, which means that $\varphi^{\mathrm{ri}}$ and $\varphi^{\mathrm{wri}}$ do not reduce the number of states of the automaton $\mathscr{A}$, but despite this, they produce crisp-deterministic fuzzy automata which are smaller than the Nerode automaton of $\mathscr{A}$.

*Example 5.2.* Let $\mathscr{A}$ be a Boolean automaton over the two-element alphabet $X = \{x, y\}$ given by the transition graph shown in Fig. 5.6 a). The transition graphs of the Nerode automaton $\mathscr{A}_N$ and its children automaton $\mathscr{A}_N^c$, constructed by means of Algorithms 5.15 and 5.21, are represented by Fig. 5.6 b) and c). Clearly, the Nerode automaton $\mathscr{A}_N$ has 7 states, and its children automaton $\mathscr{A}_N^c$ has 6 states.

On the other hand, both the greatest right invariant fuzzy quasi-order $\varphi^{\mathrm{ri}}$ and the greatest weakly right invariant fuzzy quasi-order $\varphi^{\mathrm{wri}}$ on $\mathscr{A}$, computed using Algorithms 5.17 and 5.19, are equal to the equality relation on the set of states of $\mathscr{A}$, so both automata $\mathscr{A}_{\varphi^{\mathrm{ri}}}$ and $\mathscr{A}_{\varphi^{\mathrm{wri}}}$ are isomorphic to $\mathscr{A}_N$. Therefore, we have that $|\mathscr{A}_N^c| < |\mathscr{A}_N| = |\mathscr{A}_{\varphi^{\mathrm{ri}}}| = |\mathscr{A}_{\varphi^{\mathrm{wri}}}|$, which demonstrates that construction of the children automaton, applied to the Nerode automa-

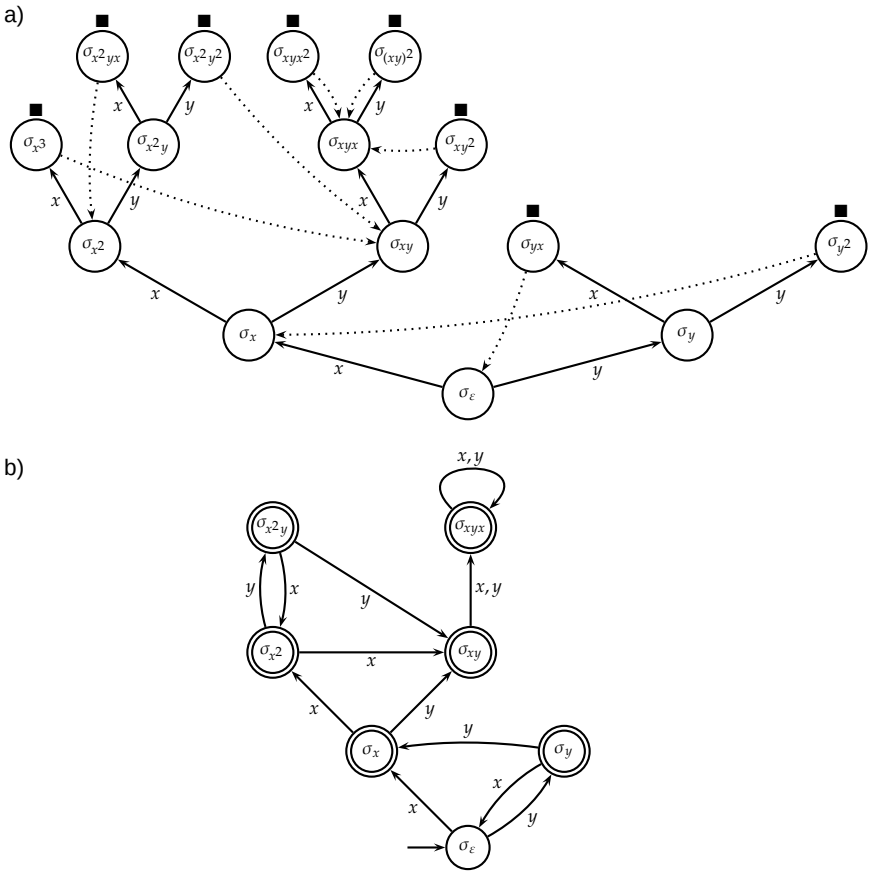



**Fig. 5.2** The transition tree (a)) and the transition graph (b)) of the Nerode automaton $\mathscr{A}_N$ of the fuzzy automaton $\mathscr{A}$ from Example 5.1.

ton of $\mathscr{A}$, can give better results in determinization of $\mathscr{A}$ than constructions based on the greatest right invariant and weakly right invariant fuzzy quasi-orders on $\mathscr{A}$.

*Example 5.3.* Let $\mathscr{A}$ be a Boolean automaton over the two-element alphabet $X = \{x, y\}$ given by the transition graph shown in Fig. 5.7 a). The Nerode automaton $\mathscr{A}_N$ and the children automaton $\mathscr{A}_N^c$ are represented by graphs in Fig. 5.7 b) and c). Moreover, using Algorithms 5.17 and 5.19 we obtain that

$$\varphi^{\mathrm{ri}} = \varphi^{\mathrm{wri}} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix},$$

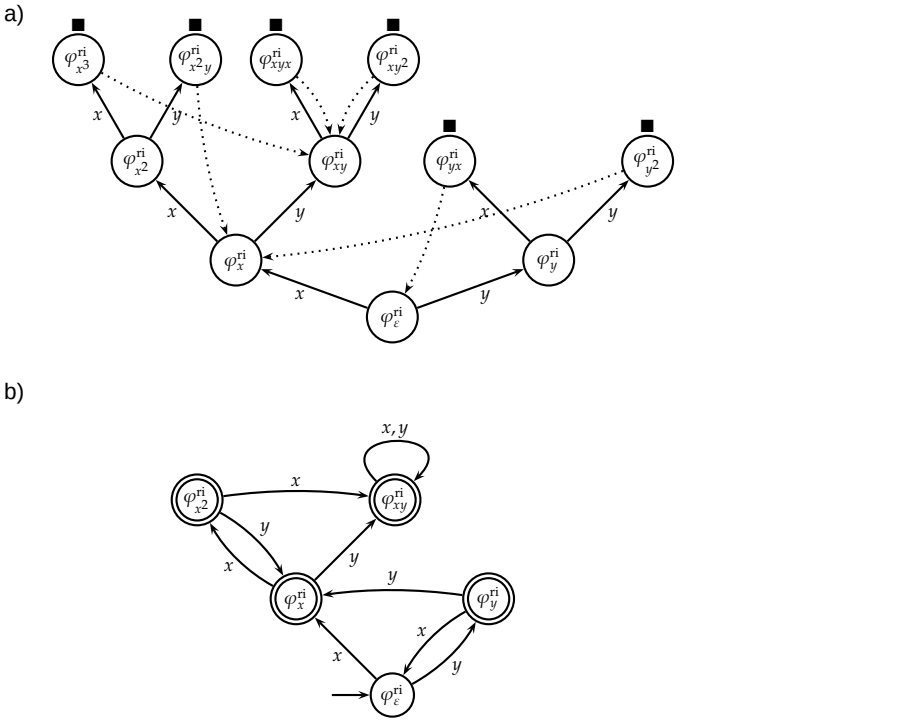


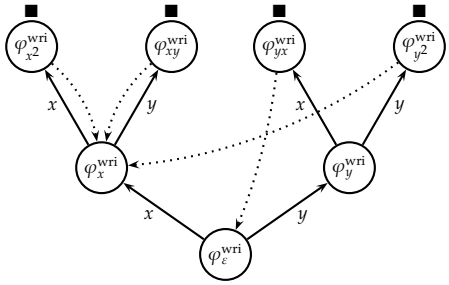

**Fig. 5.3** The transition tree (a)) and the transition graph (b)) of the automaton $\mathscr{A}_{\varphi^{\mathrm{ri}}}$ for the fuzzy automaton $\mathscr{A}$ from Example 5.1.

so automata $\mathscr{A}_{\varphi^{\mathrm{ri}}}$ and $\mathscr{A}_{\varphi^{\mathrm{wri}}}$ are mutually isomorphic and are given by the transition graph in Fig. 5.7 d), and we conclude that $|\mathscr{A}_{\varphi^{\mathrm{ri}}}| = |\mathscr{A}_{\varphi^{\mathrm{wri}}}| < |\mathscr{A}_N^c| < |\mathscr{A}_N|$.

Thus, in contrast to the previous one, this example shows that there are cases where determinization of a fuzzy automaton $\mathscr{A}$ by means of the greatest right invariant and weakly right invariant fuzzy quasi-orders can give better results than construction of the children automaton of the Nerode automaton of $\mathscr{A}$.

*Example 5.4.* Let $\mathscr{A}$ be a Boolean automaton over the two-element alphabet $X = \{x, y\}$ given by the transition graph shown in Fig. 5.8 a). When we compute $\varphi^{\mathrm{ri}}$ and $\varphi^{\mathrm{wri}}$ we obtain that both of them are equal to the equality relation on the set of states of $\mathscr{A}$, and therefore, both automata $\mathscr{A}_{\varphi^{\mathrm{ri}}}$ and $\mathscr{A}_{\varphi^{\mathrm{wri}}}$ are isomorphic to the Nerode automaton $\mathscr{A}_N$, which is represented by the transition graph in Fig. 5.8 b). Moreover, we obtain that the children automaton $\mathscr{A}_N^c$ of $\mathscr{A}_N$ is also isomorphic to the Nerode automaton $\mathscr{A}_N$. Hence, in this case none of the methods discussed in this paper does not give an automaton with smaller number of states than the Nerode automaton $\mathscr{A}_N$. It

a)




b)

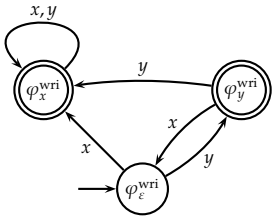



**Fig. 5.4** The transition tree (a)) and the transition graph (b)) of the automaton $\mathscr{A}_{\varphi^{\mathrm{wri}}}$ for the fuzzy automaton $\mathscr{A}$ from Example 5.1.

should be noted that the Nerode automaton $\mathscr{A}_N$ is not minimal, the minimal deterministic automaton equivalent to $\mathscr{A}_N$ is represented by the graph shown in Fig. 5.8 c).

*Example 5.5.* Let $\mathscr{A}$ be a Boolean automaton over the two-element alphabet $X = \{x, y\}$ given by the transition graph shown in Fig. 5.9 a). The transition graph of the Nerode automaton $\mathscr{A}_N$ of $\mathscr{A}$ is given in Fig. 5.9 b). Using Algorithms 5.17 and 5.19 we obtain that

$$\varphi^{\mathrm{ri}} = \varphi^{\mathrm{wri}} = \begin{bmatrix} 1&1&0&0&0 \\ 1&1&0&0&0 \\ 0&0&1&0&0 \\ 0&0&0&1&0 \\ 0&0&1&0&1 \end{bmatrix},$$

and we can easily show that $\mathscr{A}_{\varphi^{\mathrm{ri}}}$ is isomorphic to the Nerode automaton $\mathscr{A}_N$. Note that $\varphi^{\mathrm{ri}}$ has 4 different aftersets, which means that the afterset fuzzy automaton of $\mathscr{A}$ with respect to $\varphi^{\mathrm{ri}}$ has 4 states. Therefore, although $\varphi^{\mathrm{ri}}$ reduces the number of states of $\mathscr{A}$, it does not give an automaton with smaller number of states than the Nerode automaton $\mathscr{A}_N$.

*Example 5.6.* Let $\mathscr{A}$ be an automaton over the one-element alphabet $X = \{x\}$ and the Goguen (product) structure given by the transition graph shown in Fig. 5.10 a).

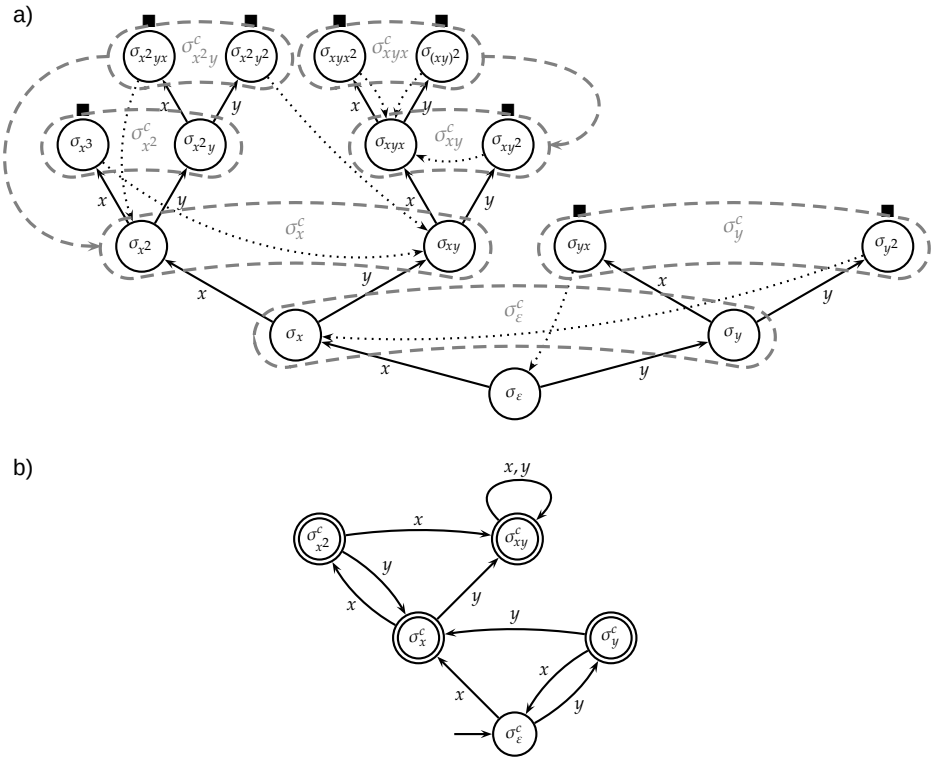



**Fig. 5.5** Construction of the transition graph of the children automaton $\mathscr{A}_N^c$ from the transition tree of the Nerode automaton $\mathscr{A}_N$ from Fig. 5.2.

It is easy to check that $\sigma_\varepsilon = [1\ 0\ 0]$, $\sigma_x = [0\ 0.5\ 1]$, and $\sigma_{x^n} = [0\ 1\ 0.5^{n-1}]$, for each $n \in \mathbb{N}$, $n \geqslant 2$. Therefore, the Nerode automaton of $\mathscr{A}$ has infinitely many states.

On the other hand, using Algorithms 5.17 and 5.19 we obtain that

$$\varphi^{\mathrm{ri}} = \varphi^{\mathrm{wri}} = \begin{bmatrix} 1 & 0 & 0.5 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix},$$

and we construct the automaton $\mathscr{A}_{\varphi^{\mathrm{ri}}}$ which is shown in Fig. 5.10 b).

Therefore, although the Nerode automaton of $\mathscr{A}$ is inifinite, using the greatest right invariant fuzzy quasi-order on $\mathscr{A}$ we obtain a finite crisp-deterministic fuzzy automaton which is equivalent to $\mathscr{A}$.

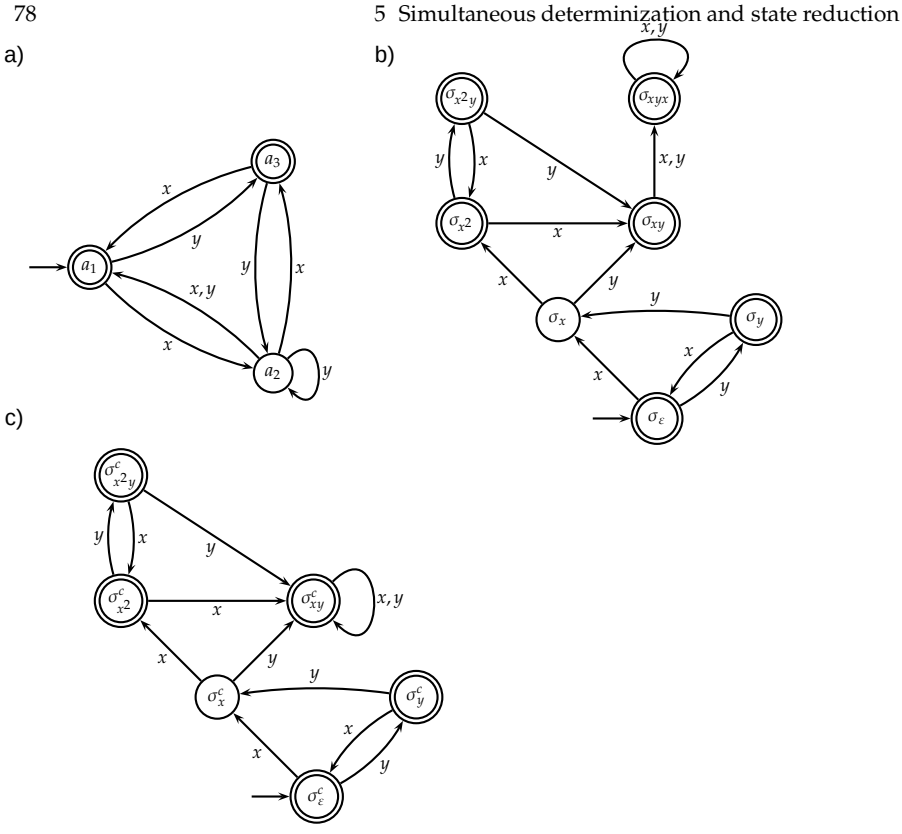



**Fig. 5.6** Transition graphs of the fuzzy automaton $\mathscr{A}$ from Example 5.2 (a)), its Nerode automaton $\mathscr{A}_N$ (b)), and the children automaton $\mathscr{A}^c_N$ (c)) .

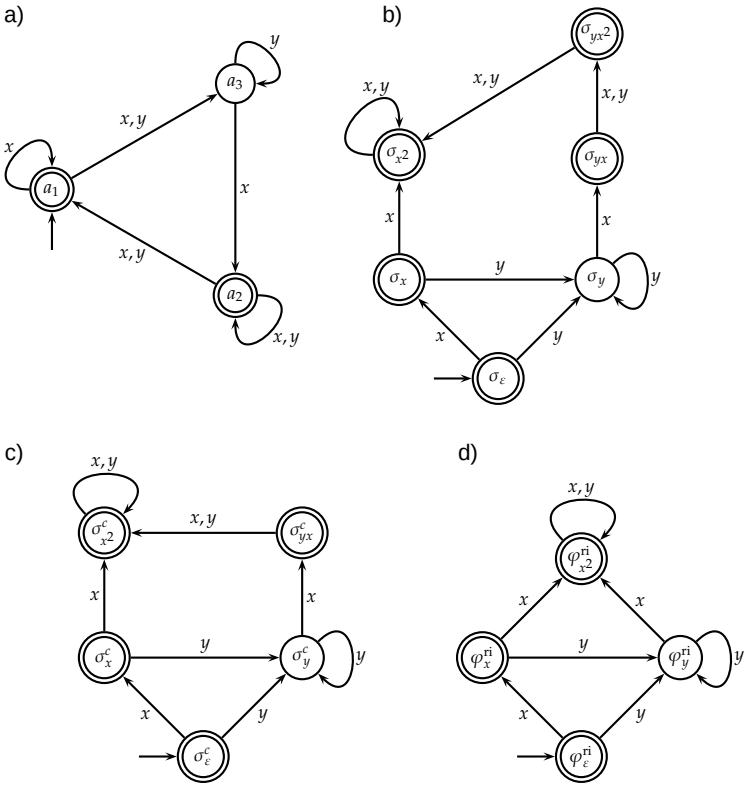



**Fig. 5.7** Transition graphs of the fuzzy automaton $\mathscr{A}$ from Example 5.3 (a)), its Nerode automaton $\mathscr{A}_N$ (b)), the children automaton $\mathscr{A}_N^{\mathrm{c}}$ of $\mathscr{A}_N$ (c)), and the automaton $\mathscr{A}_{\varphi^{\mathrm{ri}}} \cong \mathscr{A}_{\varphi^{\mathrm{wri}}}$ (d)).

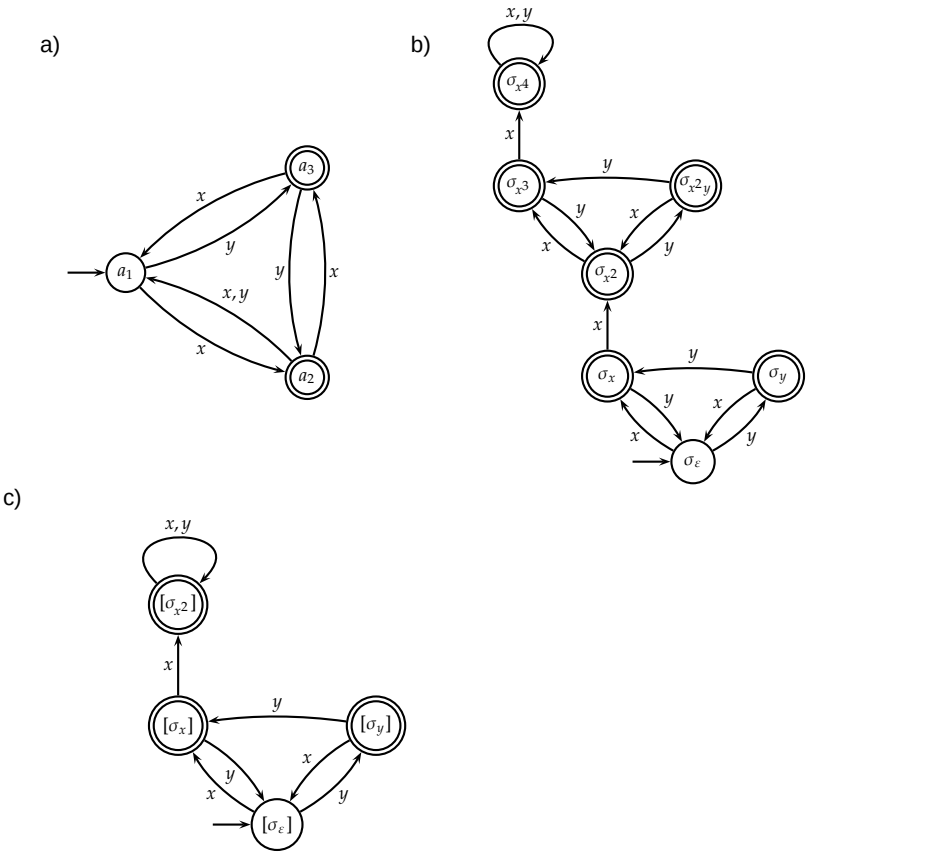



**Fig. 5.8** Transition graphs of the fuzzy automaton $\mathscr{A}$ from Example 5.4 (a)), its Nerode automaton $\mathscr{A}_N$ (b)), and the minimal crisp-deterministic fuzzy automaton equivalent to $\mathscr{A}_N$ (c)) .

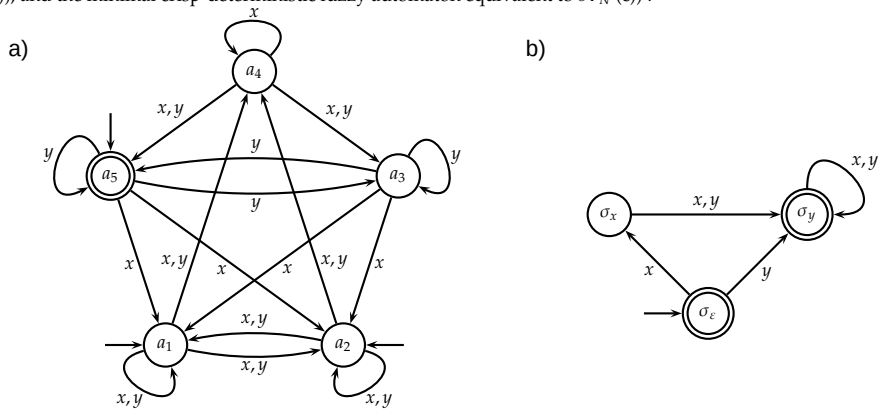



**Fig. 5.9** The transition graph of the fuzzy automaton from Example 5.5 (a)), and its Nerode automaton $\mathscr{A}_N$ (b)).

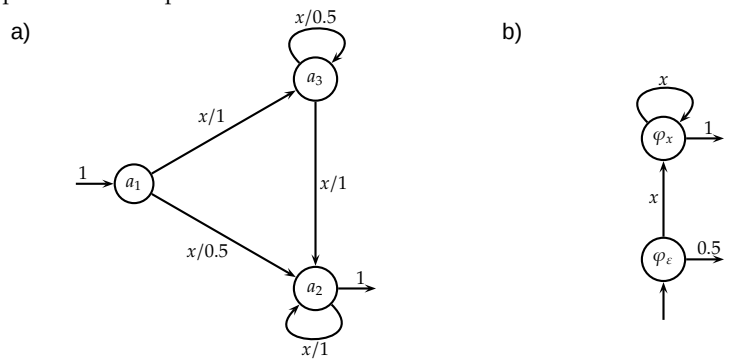



**Fig. 5.10** Transition graphs of the fuzzy automaton $\mathscr{A}$ from Example 5.6 (a)), and the automaton $\mathscr{A}_{\varphi^{\mathrm{ri}}}$ (b)).

# Chapter 6
# Determinization by means of the degree of language inclusion

Recall that canonization is referred to as such determinization method that always results in a minimal deterministic automaton equivalent to the automaton which is determinized. In Chapter 4 we have presented one canonization method for fuzzy automata, the Brzozowski type determinization. The main objective of this chapter is to provide another canonization method for fuzzy automata over a complete residuated lattice, the so-called determinization by means of the degree of language inclusion. Although the first step in this canonization procedure is substantially the same as the first step in the Brzozowski type procedure (it can be understood as the construction of the reverse Nerode automaton), these two procedures are different. If the basic operations in the underlying complete residuated lattice can be computed in constant time, the determinization by means of the degree of language inclusion has the same computational time as all other determinization algorithms considered in this thesis, including the Brzozowski type determinization.

The chapter is organized as follows. In the first section we generalize the concept of the right language associated with a state of a fuzzy automaton introducing the concept of the right language associated with a fuzzy set of states, and define the concept of the degree of language inclusion for these fuzzy languages. In Section 6.2., for a given fuzzy automaton $\mathscr{A}$, we construct an automaton $\mathscr{A}_d$, and prove that it is a minimal crisp-deterministic fuzzy automaton equivalent to $\mathscr{A}$ (Theorems 6.2 and 6.3). Finally, in Section 6.3. we provide an algorithm which constructs the automaton $\mathscr{A}_d$, analyze its computational time, and give a computational example that demonstrates the application of the algorithm.

Note that the algorithm presented in this chapter generalizes the so-called determinization using closures provided by van Glabbeek and Ploeger in [120, 121] for ordinary nondeterministic automata.

## 6.1. The degree of language inclusion

In this section we introduce the concept of a right fuzzy languages associated with fuzzy subsets of the set of states of a given fuzzy automaton, as well as the concept of the degree of language inclusion for two such fuzzy languages.

Let $\mathscr{A} = (A, \delta, \sigma, \tau)$ be a fuzzy automaton over an alphabet $X$ and a complete residuated lattice $\mathscr{L}$. Recall that the right fuzzy language associated with a state $a \in A$ is a fuzzy language $\tau_a \in \mathscr{F}(X^*)$ defined by:

$$\tau_a(u) = \bigvee_{b \in A} \delta^*(a, u, b) \otimes \tau(b),$$

for all $u \in X^*$, i.e., $\tau_a$ is the fuzzy language recognized by a fuzzy automaton $\mathscr{A}_a = (A, \delta, a, \tau)$ obtained from $\mathscr{A}$ by replacing $\sigma$ with the single crisp initial state $a$. Analogously, we can define the *right fuzzy language* associated with a fuzzy set $\mu \in \mathscr{F}(A)$ as a fuzzy language $\tau_\mu \in \mathscr{F}(X^*)$ given by

$$\tau_\mu(u) = \bigvee_{a,b \in A} \mu(a) \otimes \delta^*(a, u, b) \otimes \tau(b) = \mu \circ \tau_u,$$

for all $u \in X^*$. In other words, $\tau_\mu$ is a fuzzy language recognized by a fuzzy automaton $\mathscr{A}_\mu = (A, \delta, \mu, \tau)$ obtained from $\mathscr{A}$ by replacing $\sigma$ with $\mu$, and clearly, $\tau_\sigma = [\![\mathscr{A}]\!]$.

Recall that the degree of inclusion $I(\mu, \nu)$ of two fuzzy sets $\mu, \nu \in \mathscr{F}(A)$, in that order, is defined as follows

$$I(\mu, \nu) = \bigwedge_{a \in A} \mu(a) \to \nu(a).$$

In other words, $I(\mu, \nu)$ is a measure of "how much $\mu$ is contained in $\nu$ ".

For a fuzzy automaton $\mathscr{A} = (A, \delta, \sigma, \tau)$ and fuzzy sets $\mu, \nu \in \mathscr{F}(A)$, the degree of inclusion of fuzzy languages $\tau_\mu$ and $\tau_\nu$ is called the *degree of language inclusion* of $\tau_\mu$ and $\tau_\nu$ and denoted by $L(\mu, \nu)$, i.e.,

$$L(\mu, \nu) = \bigwedge_{u \in X^*} \tau_\mu(u) \to \tau_\nu(u).$$

In particular, if $\mu$ is a crisp singleton, i.e., $\mu = \{a\}$, for some $a \in A$, then we write $L(a, \nu)$ instead of $L(\mu, \nu)$, i.e.,

$$L(a, \nu) = \bigwedge_{u \in X^*} \tau_a(u) \to \tau_\nu(u)$$

(and similarly if $\nu$ is a crisp singleton).

The main objective of this chapter is to show that the degree of language inclusion plays a very important role in the determinization of fuzzy finite

automata, and that the determinization method based on it always results in a minimal crisp-deterministic fuzzy automaton equivalent to the original fuzzy automaton.

## 6.2. The automaton constructed by means of the degree of language inclusion

Let $\mathscr{A} = (A, \delta, \sigma, \tau)$ be a fuzzy automaton over an alphabet $X$ and a complete residuated lattice $\mathscr{L}$. For each $u \in X^*$ we define a fuzzy set $d_u : A \to L$ inductively, as follows: for the empty word $\varepsilon$ and all $u \in X^*$ and $x \in X$ we set

$$d_\varepsilon(a) = L(a, \sigma) = \bigwedge_{v \in X^*} \tau_v(a) \to \sigma \circ \tau_v, \tag{6.1}$$

and

$$d_{ux}(a) = L(a, d_u \circ \delta_x) = \bigwedge_{v \in X^*} \tau_v(a) \to d_u \circ \delta_x \circ \tau_v, \tag{6.2}$$

for all $a \in A$.

The following theorem is the most important result of this chapter.

**Theorem 6.1.** *Let $\mathscr{A} = (A, \delta, \sigma, \tau)$ be a fuzzy automaton over an alphabet $X$ and a complete residuated lattice $\mathscr{L}$. Then*

$$d_u \circ \tau_w = \sigma_u \circ \tau_w, \tag{6.3}$$

*for all $u, w \in X^*$.*

*Proof.* By induction on the length of the word $u$ we will prove that (6.3) is true for every $w \in X^*$.

For the empty word $\varepsilon$ we have that

$$d_\varepsilon(a) = \bigwedge_{v \in X^*} \tau_v(a) \to \sigma \circ \tau_v \leqslant \tau_w(a) \to \sigma \circ \tau_w$$

holds for all $a \in A$ and $w \in X^*$. Then we have

$$d_\varepsilon \circ \tau_w = \bigvee_{a \in A} d_\varepsilon(a) \otimes \tau_w(a) \leqslant \bigvee_{a \in A} (\tau_w(a) \to \sigma \circ \tau_w) \otimes \tau_w(a) \leqslant \sigma \circ \tau_w,$$

because $(p \to q) \otimes p \leqslant q$ holds for all $p, q \in L$. Therefore, we have

$$d_\varepsilon \circ \tau_w \leqslant \sigma \circ \tau_w, \tag{6.4}$$

for all $w \in X^*$, and hence we need to prove that $d_\varepsilon \circ \tau_w \geqslant \sigma \circ \tau_w$ holds for all $w \in X^*$. According to the fact that $\to$ is isotone in the second argument and $p \to (q \otimes p) \geqslant q$ we have that

$$d_\varepsilon(a) = \bigwedge_{v\in X^*} \tau_v(a) \to \sigma\circ\tau_v = \bigwedge_{v\in X^*} (\tau_v(a) \to (\bigvee_{b\in X^*} \sigma(b)\otimes\tau_v(b)))$$
$$\geqslant \bigwedge_{v\in X^*} \tau_v(a) \to (\sigma(a)\otimes\tau_v(a)) \geqslant \sigma(a),$$

for all $a \in A$, so

$$d_\varepsilon \circ \tau_w = \bigvee_{a\in A} d_\varepsilon(a)\otimes\tau_w(a) \geqslant \bigvee_{a\in A} \sigma(a)\otimes\tau_w(a) = \sigma\circ\tau_w. \tag{6.5}$$

From (6.4) and (6.5) we have that

$$d_\varepsilon \circ \tau_w = \sigma\circ\tau_w,$$

for all $w \in X^*$. Therefore, (6.3) holds for $u = \varepsilon$ and every $w \in X^*$.

Suppose that (6.3) is true for every word $u$ of length $l$, for some $l \in \mathbb{N}$, and every $w \in X^*$, and consider an arbitrary $x \in X$.

Then we have that

$$d_{ux}(a) = \bigwedge_{v\in X^*} \tau_v(a) \to d_u\circ\delta_x\circ\tau_v \leqslant \tau_w(a) \to d_u\circ\delta_x\circ\tau_w,$$

for all $a \in A$ and $w \in X^*$, and hence

$$d_{ux}\circ\tau_w = \bigvee_{a\in A} d_{ux}(a)\otimes\tau_w(a) \leqslant \bigvee_{a\in A} (\tau_w(a) \to d_u\circ\delta_x\circ\tau_w)\otimes\tau_w(a) \leqslant d_u\circ\delta_x\circ\tau_w,$$

because $(p \to q)\otimes p \leqslant q$, for all $p, q \in L$. Therefore, we have

$$d_{ux}\circ\tau_w \leqslant d_u\circ\delta_x\circ\tau_w = d_u\circ\tau_{xw} = \sigma_u\circ\tau_{xw} = \sigma_{ux}\circ\tau_w, \tag{6.6}$$

for all $w \in X^*$, so we only need to show that $d_u\circ\tau_w \geqslant \sigma_u\circ\tau_w$.

According to the fact that the operation $\to$ is isotonic in the second argument and $p \to (q\otimes p) \geqslant q$, we have that

$$d_{ux}(a) = \bigwedge_{v\in X^*} \tau_v(a) \to d_u\circ\delta_x\circ\tau_v = \bigwedge_{v\in X^*} (\tau_v(a) \to (\bigvee_{b\in X^*} (d_u\circ\delta_x)(b)\otimes\tau_v(b)))$$
$$\geqslant \bigwedge_{v\in X^*} \tau_v(a) \to ((d_u\circ\delta_x)(a)\otimes\tau_v(a)) \geqslant (d_u\circ\delta_x)(a),$$

for all $a \in A$, so

$$d_{ux}\circ\tau_w = \bigvee_{a\in A} d_{ux}(a)\otimes\tau_w(a) \geqslant \bigvee_{a\in A} (d_u\circ\delta_x)(a)\otimes\tau_w(a)$$
$$= d_u\circ\delta_x\circ\tau_w = d_u\circ\tau_{xw} = \sigma_u\circ\tau_{xw} = \sigma_{ux}\circ\tau_w.$$

From (6.6) and (6.2.) we have

$$d_u \circ \tau_w = \sigma_u \circ \tau_w,$$

for all $w \in X^*$. □

According to previous theorem we have

$$d_\varepsilon(a) = \bigwedge_{v \in X^*} \tau_v(a) \to \sigma \circ \tau_v$$

and

$$d_{ux}(a) = \bigwedge_{v \in X^*} \tau_v(a) \to d_u \circ \delta_x \circ \tau_v = \bigwedge_{v \in X^*} \tau_v(a) \to \sigma_{ux} \circ \tau_v$$

for all $a \in A$, $u \in X^*$ and $x \in X$, and hence

$$d_u(a) = \bigwedge_{v \in X^*} \tau_v(a) \to \sigma_u \circ \tau_v \tag{6.7}$$

holds for all $u \in X^*$.

Now, we set $A_d = \{d_u \mid u \in X^*\}$, and we define functions $\delta_d : A_d \times X \to A_d$ and $\tau_d : A_d \to L$ by

$$\delta_d(d_u, x) = d_{ux}, \qquad \tau_d(d_u) = d_u \circ \tau,$$

for all $d_u \in A_d$ and $x \in X$.

We have the following:

**Theorem 6.2.** *Let $\mathscr{A} = (A, \delta, \sigma, \tau)$ be a fuzzy automaton over an alphabet $X$ and a complete residuated lattice $\mathscr{L}$. Then $\mathscr{A}_d = (A_d, \delta_d, d_\varepsilon, \tau_d)$ is an accessible crisp-deterministic fuzzy automaton, and it is equivalent to $\mathscr{A}$, i.e. $[\![\mathscr{A}_d]\!] = [\![\mathscr{A}]\!]$.*

*Proof.* Let $u, v \in X^*$ such that $d_u = d_v$. Then for every $x \in X$ we have that

$$d_{ux} = \bigwedge_{w \in X^*} \tau_w(a) \to d_u \circ \delta_x \circ \tau_w = \bigwedge_{w \in X^*} \tau_w(a) \to d_v \circ \delta_x \circ \tau_w = d_{vx},$$

and hence, $\delta_d$ is a well-defined mapping. It is clear that $\tau_d$ is also a well-defined mapping. Therefore, we have that $\mathscr{A}_d = (A_d, \delta_d, d_e, \tau_d)$ is an accessible crisp-deterministic fuzzy automaton. According to the definition of the fuzzy language recognized by a fuzzy automaton, the definition of the fuzzy language recognized by a crisp-deterministic fuzzy automaton and Theorem 6.1 we have:

$$[\![\mathscr{A}_d]\!](u) = \tau_d\Big(\delta_d^*(d_e, u)\Big) = \tau_d(d_u) = d_u \circ \tau = \sigma_u \circ \tau = [\![\mathscr{A}]\!](u),$$

for every $u \in X^*$, and we have proved that $\mathscr{A}_d$ is equivalent to $\mathscr{A}$. □

**Theorem 6.3.** *Let $\mathscr{A} = (A, \delta, \sigma, \tau)$ be a fuzzy automaton over an alphabet $X$ and a complete residuated lattice $\mathscr{L}$. Then, $\mathscr{A}_d$ is a minimal crisp-deterministic fuzzy automaton equivalent to $\mathscr{A}$.*

*Proof.* Set $[\![\mathscr{A}]\!] = f$. As it has been stated in Section 2.3., the derivative automaton $\mathscr{A}_f$ of $f$ is a minimal crisp-deterministic fuzzy automaton recognizing $f$, i.e., a minimal crisp-deterministic fuzzy automaton equivalent to $\mathscr{A}$. Therefore, in order to show that the automaton $\mathscr{A}_d = (A_d, \delta_d, d_\varepsilon, \tau_d)$ is a minimal crisp-deterministic fuzzy automaton equivalent to $\mathscr{A}$ it is enough to prove that there exists a surjective mapping from $\mathscr{A}_d$ to $\mathscr{A}_f$.

Let $\phi : A_f \to A_d$ be a mapping defined by $\phi(f u^{-1}) = d_u$. By (6.7), given $u, v \in X^*$, such that $f u^{-1} = f v^{-1}$, we have

$$\begin{aligned}\sigma_u \circ \tau_w = [\![\mathscr{A}]\!](uw) = [\![\mathscr{A}]\!] u^{-1}(w) = f u^{-1}(w) = \\ = f v^{-1}(w) = [\![\mathscr{A}]\!] v^{-1}(w) = [\![\mathscr{A}]\!](vw) = \sigma_v \circ \tau_w,\end{aligned}$$

for every $w \in X^*$. Thereby,

$$d_u(a) = \bigwedge_{w \in X^*} \tau_w(a) \to \sigma_u \circ \tau_w = \bigwedge_{w \in X^*} \tau_w(a) \to \sigma_v \circ \tau_w = d_v(a)$$

for every $a \in A$. Accordingly, $\phi$ is a well defined mapping. It is easy to see that $\phi$ is surjective. Thus, the automaton $\mathscr{A}_d$ is a minimal crisp-deterministic fuzzy automaton equivalent to $\mathscr{A}$. □

## 6.3. Algorithm and a computational example

Note that in order to compute the members of the family of fuzzy sets $\{d_u\}_{u \in X^*}$, using formulas (6.1) and (6.2), we first need to compute all members of the family $\{\tau_v\}_{u \in X^*}$, what is nothing but the construction of the reverse Nerode automaton of the fuzzy automaton which we want to determinize. Therefore, the first step in the construction of the automaton $\mathscr{A}_d$ for a fuzzy automaton $\mathscr{A}$ is the construction of the reverse Nerode automaton of $\mathscr{A}$.

We provide the following algorithm.

**Algorithm 6.4 (*Construction of the automaton* $\mathscr{A}_d$)** The input of this algorithm is a fuzzy finite automaton $\mathscr{A} = (A, \delta, \sigma, \tau)$ with $n$ states, over a finite alphabet $X$ with $m$ letters and a complete residuated lattice $\mathscr{L}$, and the output is the crisp-deterministic automaton $\mathscr{A}_d = (A_d, \delta_d, d_\varepsilon, \tau_d)$.

The procedure is to construct the *transition tree* of $\mathscr{A}_d$ directly from $\mathscr{A}$, and during this procedure we use pointers $s(\cdot)$ which points vertices of the tree under construction to the corresponding integers. The transition tree of $\mathscr{A}_d$ is constructed inductively as follows:

(A1) First we compute all members of the family $\{\tau_v\}_{u \in X^*}$, by means of Algorithm 4.5.

(A2) The root of the tree is $d_\varepsilon$, computed using formula (6.1), and we put $T_0 = \{d_\varepsilon\}$.

(A3) After the $i$th step let a tree $T_i$ have been constructed, and vertices in $T_i$ have been labelled either 'closed' or 'non-closed'. The meaning of these two terms will be made clear in the sequel.

(A4) In the next step we construct a tree $T_{i+1}$ by enriching $T_i$ in the following way: for any non-closed leaf $d_u$ occuring in $T_i$, where $u \in X^*$, and each $x \in X$ we add a vertex $d_{ux}$ computed using formula (6.2), and an edge from $d_u$ to $d_{ux}$ labelled by $x$. Simultaneously, we check whether $d_{ux}$ is a fuzzy set that has already been constructed. If it is true, if $d_{ux}$ is equal to some previously computed $d_v$, we mark $d_{ux}$ as closed and set $s(d_{ux}) = s(d_v)$. Otherwise, we compute the value $\tau_d(d_u) = d_u \circ \tau$ and set $s(d_{ux})$ to be the next unassigned integer. The procedure terminates when all leaves are marked closed.

(A5) When the transition tree of $\mathscr{A}_d$ is constructed, we erase all closure marks and glue leaves to interior vertices with the same pointer value. The diagram that results is the transition graph of $\mathscr{A}_d$.

Let $\mathscr{A} = (A, \delta, \sigma, \tau)$ be a fuzzy finite automaton with $n$ states and $m$ input letters, and suppose that the subsemiring $\mathscr{L}^*(\delta, \sigma, \tau)$ of the semiring $(L, \vee, \otimes, 0, 1)$ generated by all membership values taken by $\delta$, $\sigma$ and $\tau$ is finite and has $k$ elements. As we have seen, the step (A1) produces an automaton with at most $k^n$ states, and the computational time of this step is $O(mnk^{2n})$. Subsequent steps produce the transition tree of a minimal crisp-deterministic fuzzy automaton equivalent to $\mathscr{A}$, which can not be larger than the Nerode automaton $\mathscr{A}_N$ of $\mathscr{A}$. As the Nerode automaton $\mathscr{A}_N$ can not have more than $k^n$ states, the resulting transition tree for $\mathscr{A}_d$ can not have more than $k^n$ internal vertices, and the total number of vertices does not exceed $mk^n + 1$. In contrast to other algorithms presented in this thesis, where the most time-demanding part is the check whether the just computed fuzzy set is a copy of some previously computed fuzzy set, here most of the time is spent on computing the fuzzy sets $d_u$, $u \in X^*$. Namely, we can write (6.1) as

$$d_\varepsilon(a) = \bigwedge_{\mu \in A_{\overline{N}}} \mu(a) \to \sigma \circ \mu, \tag{6.8}$$

for all $a \in A$, and (6.2) as

$$d_{ux}(a) = \bigwedge_{\mu \in A_{\overline{N}}} \mu(a) \to d_u \circ \mu_x, \tag{6.9}$$

for all $a \in A$, $u \in X^*$ and $x \in X$, where $\mu_x$ denotes the $x$-child of $\mu$ in the transition tree of the reverse Nerode automaton $\mathscr{A}_{\overline{N}}$ of $\mathscr{A}$ (if $\mu = \tau_v$, for some $v \in X^*$, then $\mu_x = \tau_{xv}$). The computation of any single $d_u \circ \mu_x$ (and, similarly, of $\sigma \circ \mu$) takes $O(n(c_\otimes + c_\vee))$ time, and therefore, the computation of the whole collection of values $\{d_u \circ \mu_x\}_{\mu \in A_{\overline{N}}}$ takes $O(nk^n(c_\otimes + c_\vee))$ time, since $|A_{\overline{N}}| \leqslant k^n$. When all values $d_u \circ \mu_x$ are stored, then the computation of $d_{ux}(a)$ requires time $O(k^n(c_\to + c_\wedge))$, and the computation of $d_{ux}$ requires time $O(nk^n(c_\to + c_\wedge))$. Hence, the compu-

tation of any single vertice of the transition tree of the automaton $\mathscr{A}_d$ requires time $O(nk^n(c_\otimes + c_\vee + c_\to + c_\wedge))$, and since there are at most $mn^k + 1$ vertices to be computed, the computation of all verices requires $O(mnk^{2n}(c_\otimes + c_\vee + c_\to + c_\wedge))$ time. As for the other algorithms considered in this thesis, the check whether the just computed fuzzy set is a copy of some previously computed fuzzy set is performed in time $O(mnk^{2n})$. Therefore, the whole algorithm runs in time $O(mnk^{2n}(c_\otimes + c_\vee + c_\to + c_\wedge))$, and if the operations in $\mathscr{L}$ can be computed in constant time, then this algorithm has the computational time $O(mnk^{2n})$, the same as other determinization algorithms presented here.

Now we give a computational example that demonstrates the application of Algorithm 6.4.

*Example 6.1.* Let $\mathscr{A} = (A, \delta, \sigma, \tau)$ be a fuzzy automaton over the alphabet $X = \{x\}$ and the Boolean lattice $\mathbb{B}$ given by the transition graph shown in Figure 6.1. Then $\sigma$, $\delta_x$ and $\tau$ are represented as follows:

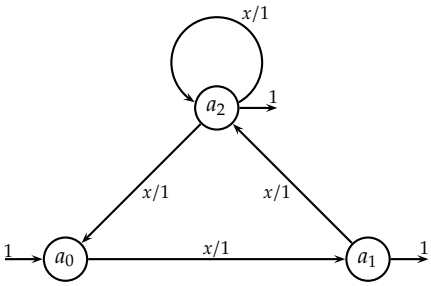



**Fig. 6.1** The transition graph of the fuzzy automaton $\mathscr{A}$

$$\sigma = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}, \quad \delta_x = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 1 \end{bmatrix}, \quad \tau = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}.$$

Fuzzy sets $\tau_u : A \to L$ computed using the Algorithm 4.5 are given by

$$\tau_\varepsilon = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}, \quad \tau_u = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix},$$

where $\varepsilon \in X^*$ denotes an empty word and $u \in X^+$. Hence, fuzzy sets $d_u \in \mathscr{F}(A)$ are given by:

$$d_\varepsilon(a_0) = \Big(\tau_e(a_0) \to \Big(\begin{bmatrix}1&0&0\end{bmatrix}\cdot\begin{bmatrix}0\\1\\1\end{bmatrix}\Big)\Big) \wedge \Big(\tau_x(a_0) \to \Big(\begin{bmatrix}1&0&0\end{bmatrix}\cdot\begin{bmatrix}1\\1\\1\end{bmatrix}\Big)\Big) = 1,$$

$$d_e(a_1) = \Big(\tau_e(a_1) \to \Big(\begin{bmatrix}1&0&0\end{bmatrix}\cdot\begin{bmatrix}0\\1\\1\end{bmatrix}\Big)\Big) \wedge \Big(\tau_x(a_1) \to \Big(\begin{bmatrix}1&0&0\end{bmatrix}\cdot\begin{bmatrix}1\\1\\1\end{bmatrix}\Big)\Big) = 0,$$

$$d_e(a_2) = \Big(\tau_e(a_2) \to \Big(\begin{bmatrix}1&0&0\end{bmatrix}\cdot\begin{bmatrix}0\\1\\1\end{bmatrix}\Big)\Big) \wedge \Big(\tau_x(a_2) \to \Big(\begin{bmatrix}1&0&0\end{bmatrix}\cdot\begin{bmatrix}1\\1\\1\end{bmatrix}\Big)\Big) = 0,$$

that is,

$$d_\varepsilon = \begin{bmatrix}1&0&0\end{bmatrix}.$$

Next, for $u = x$ we have:

$$d_x(a_0) = \Big(\tau_e(a_0) \to \Big(\begin{bmatrix}1&0&0\end{bmatrix}\cdot\begin{bmatrix}0&1&0\\0&0&1\\1&0&1\end{bmatrix}\cdot\begin{bmatrix}0\\1\\1\end{bmatrix}\Big)\Big) \wedge \Big(\tau_x(a_0) \to \Big(\begin{bmatrix}1&0&0\end{bmatrix}\cdot\begin{bmatrix}0&1&0\\0&0&1\\1&0&1\end{bmatrix}\cdot\begin{bmatrix}1\\1\\1\end{bmatrix}\Big)\Big) = 1,$$

$$d_x(a_1) = \Big(\tau_e(a_1) \to \Big(\begin{bmatrix}1&0&0\end{bmatrix}\cdot\begin{bmatrix}0&1&0\\0&0&1\\1&0&1\end{bmatrix}\cdot\begin{bmatrix}0\\1\\1\end{bmatrix}\Big)\Big) \wedge \Big(\tau_x(a_1) \to \Big(\begin{bmatrix}1&0&0\end{bmatrix}\cdot\begin{bmatrix}0&1&0\\0&0&1\\1&0&1\end{bmatrix}\cdot\begin{bmatrix}1\\1\\1\end{bmatrix}\Big)\Big) = 1,$$

$$d_x(a_2) = \Big(\tau_e(a_2) \to \Big(\begin{bmatrix}1&0&0\end{bmatrix}\cdot\begin{bmatrix}0&1&0\\0&0&1\\1&0&1\end{bmatrix}\cdot\begin{bmatrix}0\\1\\1\end{bmatrix}\Big)\Big) \wedge \Big(\tau_x(a_2) \to \Big(\begin{bmatrix}1&0&0\end{bmatrix}\cdot\begin{bmatrix}0&1&0\\0&0&1\\1&0&1\end{bmatrix}\cdot\begin{bmatrix}1\\1\\1\end{bmatrix}\Big)\Big) = 1,$$

that is,

$$d_x = \begin{bmatrix}1&1&1\end{bmatrix}.$$

Analogously, we have that

$$d_{xx} = \begin{bmatrix}1&1&1\end{bmatrix}.$$

Therefore, for every $u \in X^*$ such that $|u| \geqslant 1$ we have

$$d_u = \begin{bmatrix}1&1&1\end{bmatrix}.$$

Hence, $A_d = \{d_\varepsilon, d_x\}$, and the transition tree and the graph of the automaton $\mathscr{A}_d = (A_d, \delta_d, d_\varepsilon, \tau_d)$ are shown in Figure 6.2.

Further, let us consider the Nerode automaton $\mathscr{A}_N$ and the reduced Nerode automaton $\mathscr{A}_R$ of $\mathscr{A}$. It is easy to verify that

$$\sigma_\varepsilon = \begin{bmatrix}1&0&0\end{bmatrix}, \quad \sigma_x = \begin{bmatrix}0&1&0\end{bmatrix}, \quad \sigma_{x^2} = \begin{bmatrix}0&0&1\end{bmatrix}, \quad \sigma_{x^3} = \begin{bmatrix}1&0&1\end{bmatrix},$$

and

$$\sigma_u = \begin{bmatrix}1&1&1\end{bmatrix},$$

for all $u \in X^*$ such that $|u| \geqslant 4$. Therefore,

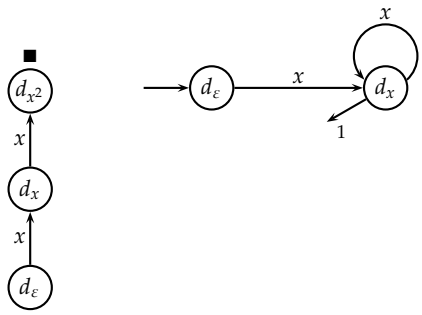



**Fig. 6.2** The transition tree and the graph of the automaton $\mathscr{A}_d$ of the fuzzy automaton $\mathscr{A}$ given in Figure 6.1.

$$\theta_\varepsilon = (\begin{bmatrix}0 & 1 & 0\end{bmatrix}, 0) \quad \theta_x = (\begin{bmatrix}0 & 0 & 1\end{bmatrix}, 1) \quad \theta_{x^2} = \begin{bmatrix}1 & 0 & 1\end{bmatrix}, 1),$$

and

$$\theta_u = (\begin{bmatrix}1 & 1 & 1\end{bmatrix}, 1),$$

for all $u \in X^*$ such that $|u| \geqslant 3$. According to this, the Nerode automaton $\mathscr{A}_N$ has 5 states, i.e. $A_N = \{\sigma_\varepsilon, \sigma_x, \sigma_{x^2}, \sigma_{x^3}, \sigma_{x^4}\}$, whereas the reduced Nerode automaton has 4 states.

# Appendix A
# C codes

## Computing the Nerode, reduced Nerode and Brzozowski's automaton

The aim of Apendix A is to present a C implementation of some of the algorithms developed in this dissertation. Namely, the input of the first program presented here, called "Brzozwski.cs", is the data describing a given fuzzy (weighted) automaton, such as number of states of automaton, size of alphabet, initial weight vector, final weight vector and weighted transition matrix, and the output is the data regarding the corresponding Nerode automaton or Brzozowski's automaton, depending on what we choose. The input of the second program, called "ReducedNerode.cs", is the data describing a given fuzzy (weighted) automaton, while the output represents the data regarding the appropriate reduced Nerode automaton. In other words, the second program is an implementation of Algorithm 3.5.

Brzozowski.c

```
#ifdef _MSC_VER
#define _CRT_SECURE_NO_WARNINGS
#endif

#include <stdio.h>   /* required for file operations */
#include <dos.h>  /* for delay */
#include <string.h>
#include <stdlib.h>

struct node {
  int* sigmau;
  int letter;
  int label;
  struct node *equalWith;
  struct node *firstchild;
  struct node *nextsibling;
};

typedef struct node *tree;

struct nodequeue {
      struct node * treenode;
      struct nodequeue *next;
};

typedef struct nodequeue * queue;

/****** INPUT DATA ************/
int n; /* number of states */
int m; /* number of input letters */
int *sigma; /* input vector */
int *tau; /* output vector */
int ***deltaArray; /* Transition matrices */
/******* END INPUT DATA ***************/

tree root = NULL;
queue q = NULL;

/******* OUTPUT DATA ************/
int nOut;
int mOut;
int *sigmaOut;
int *tauOut;
int ***deltaArrayOut;
/******* END OUTPUT DATA ********/


queue addToQueue(queue q, tree p){
      struct nodequeue * temp;
      queue t = q;
      temp = (queue) malloc(sizeof(struct nodequeue));
      temp->treenode = p;
```

Brzozowski.c

```
temp->next = NULL;
      if(q == NULL){
            q = temp;
      } else {
            while(t->next)
                  t = t->next;
            t->next = temp;
      }
      return q;
}

tree deleteFromQueue(queue *q){
      queue qtmp;
      tree tmp;
      if(*q==NULL)
            return NULL;
      tmp = (*q)->treenode;
      qtmp = (*q);
      (*q) = (*q)->next;
      free(qtmp);
      return tmp;
}

int * multiplication(int * sigma, int ** delta){
      int * result;
      int i,j;
      result = (int *)calloc(n,sizeof(int));
      for(i=0;i<n;i++){
            result[i]=0;
            for(j=0;j<n;j++){
                  result[i] = result[i] || (sigma[j] && delta[j][i]);
            }
      }
      return result;
}

void addToTree(tree parent, tree child){
      tree tmp;
      if(parent->firstchild == NULL)
            parent->firstchild = child;
      else{
            tmp = parent->firstchild;
            while(tmp->nextsibling)
                  tmp = tmp->nextsibling;
            tmp->nextsibling = child;
      }

}

void equalToTreeNode(tree tnode){
      queue qtmp;
```

Brzozowski.c

```
tree currentNode,tmpNode;
      int equal,i;
      equal = 1;
      for(i=0;i<n;i++){
            equal = equal && (tnode->sigmau[i] == root->sigmau[i]);
      }
      if(equal){
            tnode->equalWith = root;
            return;
      }
      qtmp = NULL;
      qtmp = addToQueue(qtmp,root);
      while(qtmp){
            currentNode = deleteFromQueue(&qtmp);
            equal = 1;
            for(i=0;i<n;i++){
                  equal = equal && (tnode->sigmau[i] == currentNode-
>sigmau[i]);
            }
            if(equal){
                  tnode->equalWith = currentNode;
                  return;
            }
            if(currentNode->firstchild){
                  qtmp = addToQueue(qtmp,currentNode->firstchild);
                  tmpNode = currentNode->firstchild->nextsibling;
                  while(tmpNode){
                        qtmp = addToQueue(qtmp,tmpNode);
                        tmpNode = tmpNode->nextsibling;
                  }
            }
      }
}

int scalarProduct(int *v1, int *v2, int n){
      int i, result;
      result = 0;
      for(i=0;i<n;i++)
            result = result || (v1[i] * v2[i]);
      return result;
}

void loadFromFile(char * inputFile){
      char line[80];
    char * part;
      int i,j,k;
      FILE * exampleFile;
    /* open the file for reading */
      exampleFile = fopen (inputFile, "r");

    /* reading the first line from file - n m */
```

Brzozowski.c

```
fgets(line, 80, exampleFile);
    line[ strlen(line) - 1 ] = '\0';
    /* parsing n and m */
    part = strtok(line," ");
    n = atoi(part);
    part = strtok(NULL," ");
    m = atoi(part);

    /* parsing sigma */
    sigma = (int *)calloc(n,sizeof(int));
    fgets(line, 80, exampleFile);
    line[ strlen(line) - 1 ] = '\0';
    part = strtok(line," ");
      i=0;
    while(part != NULL){
           sigma[i++] = atoi(part);
         part = strtok(NULL," ");
    }

      /* parsing tau */
    tau = (int *)calloc(n,sizeof(int));
    fgets(line, 80, exampleFile);
    line[ strlen(line) - 1 ] = '\0';
    part = strtok(line," ");
      i=0;
    while(part != NULL){
           tau[i++] = atoi(part);
         part = strtok(NULL," ");
    }

      deltaArray = (int ***)calloc(m,sizeof(int**));

      for(i=0;i<m;i++){
            deltaArray[i] = (int **)calloc(n,sizeof(int*));
            for(k=0;k<n;k++)
                  deltaArray[i][k] = (int*)calloc(n,sizeof(int));
            for(k=0;k<n;k++){
                  fgets(line, 80, exampleFile);
                  line[ strlen(line) - 1 ] = '\0';
                  part = strtok(line," ");
                  j = 0;
                  while(part != NULL){
                        deltaArray[i][k][j++] = atoi(part);
                        part = strtok(NULL," ");
                  }
            }
      }

   printf("Parsed numbers: n = %d, m = %d\n",n,m);
   for(i=0;i<n;i++)
         printf("%d ",sigma[i]);
```

Brzozowski.c

```
   printf("\n");
   for(i=0;i<n;i++)
         printf("%d ",tau[i]);
   printf("\n");

   printf("\n****************** DELTA ***************\n");
   for(i=0;i<m;i++){
         printf("\n*************** MATRIX NO. %d
*****************\n",i);
         for(j=0;j<n;j++){
               for(k=0;k<n;k++)
                     printf("%d ",deltaArray[i][j][k]);
               printf("\n");
         }
   }

   fclose(exampleFile);
}

void print(tree t){
      int i;
      printf("\n************** TREE NODE: %lu **********\n",t);
      printf("\n************** FIRST CHILD: %lu **********\n",t-
>firstchild);
      printf("\n************** NEXT SIBILING: %lu **********\n",t-
>nextsibling);
      printf("\n************** EQUAL TO: %lu **********\n",t-
>equalWith);
      printf("LETTER: %d\n",t->letter);
      printf("LABEL: %d\n",t->label);
      printf("***************** SIGMAU ************\n");
      for(i=0;i<n;i++){
            printf("%d ",t->sigmau[i]);
      }
      printf("\n");
}

void computeNerodAutomaton(){
      int i,j,k;
      tree treenode,newTreeNode,tmp;
      root = (tree) malloc(sizeof(struct node));

      //constructing the treee
      root->sigmau = (int*) calloc(n,sizeof(int));
      for(i=0;i<n;i++)
            root->sigmau[i] = sigma[i];
      root->letter = -1;
      root->equalWith = NULL;
      root->nextsibling = NULL;
      root->firstchild = NULL;
      root->label = 0;
```

Brzozowski.c

```
      q = addToQueue(q,root);
      nOut = 1;
      while(q){
            treenode = deleteFromQueue(&q);
            for(i=0;i<m;i++){
                  newTreeNode = (tree)malloc(sizeof(struct node));
                  newTreeNode->equalWith = NULL;
                  newTreeNode->firstchild = NULL;
                  newTreeNode->nextsibling = NULL;
                  newTreeNode->letter = i;
                  newTreeNode->sigmau = multiplication(treenode-
>sigmau,deltaArray[i]);
                  equalToTreeNode(newTreeNode);
                  addToTree(treenode,newTreeNode);
                  if(newTreeNode->equalWith == NULL){
                        nOut++;
                        newTreeNode->label = nOut-1;
                        q = addToQueue(q,newTreeNode);
                  } else {
                        newTreeNode->label = newTreeNode->equalWith-
>label;
                  }
            }
      }
      mOut = m;
      tauOut = (int *)calloc(nOut,sizeof(int));
      sigmaOut = (int *)calloc(nOut,sizeof(int));
      deltaArrayOut = (int ***)calloc(m,sizeof(int**));

      //constructing output delata array
      for(i=0;i<m;i++){
            deltaArrayOut[i] = (int **)calloc(nOut,sizeof(int*));
            for(k=0;k<nOut;k++)
                  deltaArrayOut[i][k] = (int*)calloc(nOut,sizeof(int));
            for(k=0;k<nOut;k++)
                  for(j=0;j<nOut;j++)
                        deltaArrayOut[i][k][j] = 0;
      }

      //walk through the tree
      q = NULL;
      q = addToQueue(q,root);
      while(q){
            treenode = deleteFromQueue(&q);
            print(treenode);
            if(treenode->equalWith == NULL){
                  tauOut[treenode->label] = scalarProduct(treenode-
>sigmau,tau,n);
            }
            if(treenode->firstchild){
                  q = addToQueue(q, treenode->firstchild);
```

Brzozowski.c

```
                tmp = treenode->firstchild->nextsibling;
                //pupulating output delta array
                deltaArrayOut[treenode->firstchild->letter][treenode-
>label][treenode->firstchild->label] = 1;
                while(tmp){
                      deltaArrayOut[tmp->letter][treenode->label][tmp-
>label] = 1;
                      q = addToQueue(q, tmp);
                      tmp = tmp->nextsibling;
                }
          }
      }

      //populating sigma array
      sigmaOut[0]=1;
      for(i=1;i<nOut;i++)
            sigmaOut[i] = 0;
}

void reverseAutomation(int *sigma, int *tau, int n, int ***deltas, int
m){
      int i,j,k, tmp;
      //exchanging sigma and tay arrays
      for(i=0;i<n;i++){
            tmp = sigma[i];
            sigma[i] = tau[i];
            tau[i] = tmp;
      }
      for(k=0;k<m;k++){
            for(i=0;i<n;i++){
                  for(j=i+1;j<n;j++){
                        tmp = deltas[k][i][j];
                        deltas[k][i][j] = deltas[k][j][i];
                        deltas[k][j][i] = tmp;
                  }
            }
      }
}

void outputDataToInputData(){
      int i,j,k;

      free(sigma);
      free(tau);
      sigma = (int *)calloc(nOut,sizeof(int));
      tau = (int *)calloc(nOut,sizeof(int));
      for(i=0;i<nOut;i++){
            sigma[i] = sigmaOut[i];
            tau[i] = tauOut[i];
      }
      free(sigmaOut);
```

Brzozowski.c

```
	free(tauOut);

	//free delta input array

	for(i=0;i<m;i++)
		for(j=0;j<n;j++)
			free(deltaArray[i][j]);
	for(i=0;i<m;i++)
		free (deltaArray[i]);
	free(deltaArray);

	deltaArray = (int ***)calloc(mOut,sizeof(int**));
	for(i=0;i<mOut;i++){
		deltaArray[i] = (int **)calloc(nOut,sizeof(int*));
		for(k=0;k<nOut;k++)
			deltaArray[i][k] = (int*)calloc(nOut,sizeof(int));
	}

	for(i=0;i<mOut;i++)
		for(j=0;j<nOut;j++)
			for(k=0;k<nOut;k++)
				deltaArray[i][j][k] = deltaArrayOut[i][j][k];

	//free delta output array

	for(i=0;i<mOut;i++)
		for(j=0;j<nOut;j++)
			free(deltaArrayOut[i][j]);
	for(i=0;i<mOut;i++)
		free (deltaArrayOut[i]);
	free(deltaArrayOut);

	n = nOut;
	m = mOut;
}

void writeOutputFile(char *outputFileName){
	int i,j,k;
	FILE *g;
	g = fopen(outputFileName,"w");
	fprintf(g,"%d %d\n",nOut,mOut);
	//write sigma array
	for(i=0;i<nOut-1;i++)
		fprintf(g,"%d ",sigmaOut[i]);
	fprintf(g,"%d\n",sigmaOut[nOut-1]);
	//write tau array
	for(i=0;i<nOut-1;i++)
		fprintf(g,"%d ",tauOut[i]);
	fprintf(g,"%d\n",tauOut[nOut-1]);

	for(i=0;i<mOut;i++){
```

Brzozowski.c

```
            for(j=0;j<nOut;j++){
                  for(k=0;k<nOut-1;k++){
                        fprintf(g,"%d ",deltaArrayOut[i][j][k]);
                  }
                  fprintf(g,"%d\n",deltaArrayOut[i][j][nOut-1]);
            }
      }
      fclose(g);
}

void main(){
      int c;
      char* inputFile = "primer2.txt";
      char* outputFile;
      loadFromFile(inputFile);

      printf("\nChoose an option:\n");
      printf("1. Compute Nerodov automation\n");
      printf("2. Compute Brzozovski automation\n");
      printf("3. Compute Reduced Nerode Automation\n");

      scanf("%d",&c);
      switch(c){
      case 1:
            computeNerodAutomaton();
            outputFile = (char *) calloc(100,sizeof(char));
            strcpy(outputFile,"result nerodov - ");
            strcat(outputFile,inputFile);
            break;
      case 2:
            reverseAutomation(sigma,tau,n,deltaArray,m);
            computeNerodAutomaton();
            outputDataToInputData();
            reverseAutomation(sigma,tau,n,deltaArray,m);
            computeNerodAutomaton();
            outputFile = (char *) calloc(100,sizeof(char));
            strcpy(outputFile,"result brzozovski - ");
            strcat(outputFile,inputFile);
            break;
      case 3:
            break;
      }
      //write result
      writeOutputFile(outputFile);
}
```

ReducedNerode.c

```
#ifdef _MSC_VER
#define _CRT_SECURE_NO_WARNINGS
#endif

#include <stdio.h>   /* required for file operations */
#include <dos.h>  /* for delay */
#include <string.h>
#include <stdlib.h>

struct node {
  int** etau;
  int kapa;
  int letter;
  int label;
  struct node *equalWith;
  struct node *firstchild;
  struct node *nextsibling;
};

typedef struct node *tree;

struct nodequeue {
      struct node * treenode;
      struct nodequeue *next;
};

typedef struct nodequeue * queue;

/****** INPUT DATA ************/
int n; /* number of states */
int m; /* number of input letters */
int *sigma; /* input vector */
int *tau; /* output vector */
int ***deltaArray; /* Transition matrices */
/******* END INPUT DATA ***************/

tree root = NULL;
queue q = NULL;

/******* OUTPUT DATA ************/
int nOut;
int mOut;
int *sigmaOut;
int *tauOut;
int ***deltaArrayOut;
/******* END OUTPUT DATA ********/


queue addToQueue(queue q, tree p){
      struct nodequeue * temp;
      queue t = q;
```

ReducedNerode.c

```
      temp = (queue) malloc(sizeof(struct nodequeue));
      temp->treenode = p;
      temp->next = NULL;
      if(q == NULL){
            q = temp;
      } else {
            while(t->next)
                  t = t->next;
            t->next = temp;
      }
      return q;
}

tree deleteFromQueue(queue *q){
      queue qtmp;
      tree tmp;
      if(*q==NULL)
            return NULL;
      tmp = (*q)->treenode;
      qtmp = (*q);
      (*q) = (*q)->next;
      free(qtmp);
      return tmp;
}

int * multiplication(int * sigma, int ** delta){
      int * result;
      int i,j;
      result = (int *)calloc(n,sizeof(int));
      for(i=0;i<n;i++){
            result[i]=0;
            for(j=0;j<n;j++){
                  result[i] = result[i] || (sigma[j] && delta[j][i]);
            }
      }
      return result;
}

void addToTree(tree parent, tree child){
      tree tmp;
      if(parent->firstchild == NULL)
            parent->firstchild = child;
      else{
            tmp = parent->firstchild;
            while(tmp->nextsibling)
                  tmp = tmp->nextsibling;
            tmp->nextsibling = child;
      }

}
```

ReducedNerode.c

```
void equalToTreeNode(tree tnode){
      queue qtmp;
      tree currentNode,tmpNode;
      int equal,i,j;
      equal = 1;
      equal = equal && (tnode->kapa == root->kapa);
      for(j=0;j<m;j++){
            for(i=0;i<n;i++){
                  equal = equal && (tnode->etau[j][i] == root-
>etau[j][i]);
            }
      }
      if(equal){
            tnode->equalWith = root;
            return;
      }
      qtmp = NULL;
      qtmp = addToQueue(qtmp,root);
      while(qtmp){
            currentNode = deleteFromQueue(&qtmp);
            equal = 1;
            equal = equal && (tnode->kapa == currentNode->kapa);
            for(j=0;j<m;j++){
                  for(i=0;i<n;i++){
                        equal = equal && (tnode->etau[j][i] ==
currentNode->etau[j][i]);
                  }
            }
            if(equal){
                  tnode->equalWith = currentNode;
                  return;
            }
            if(currentNode->firstchild){
                  qtmp = addToQueue(qtmp,currentNode->firstchild);
                  tmpNode = currentNode->firstchild->nextsibling;
                  while(tmpNode){
                        qtmp = addToQueue(qtmp,tmpNode);
                        tmpNode = tmpNode->nextsibling;
                  }
            }
      }
}

int scalarProduct(int *v1, int *v2, int n){
      int i, result;
      result = 0;
      for(i=0;i<n;i++)
            result = result || (v1[i] * v2[i]);
      return result;
}
```

ReducedNerode.c

```
void loadFromFile(char * inputFile){
      char line[80];
    char * part;
      int i,j,k;
      FILE * exampleFile;
    /* open the file for reading */
      exampleFile = fopen (inputFile, "r");

    /* reading the first line from file - n m */
    fgets(line, 80, exampleFile);
    line[ strlen(line) - 1 ] = '\0';
    /* parsing n and m */
    part = strtok(line," ");
    n = atoi(part);
    part = strtok(NULL," ");
    m = atoi(part);

    /* parsing sigma */
    sigma = (int *)calloc(n,sizeof(int));
    fgets(line, 80, exampleFile);
    line[ strlen(line) - 1 ] = '\0';
    part = strtok(line," ");
      i=0;
    while(part != NULL){
            sigma[i++] = atoi(part);
          part = strtok(NULL," ");
    }

      /* parsing tau */
    tau = (int *)calloc(n,sizeof(int));
    fgets(line, 80, exampleFile);
    line[ strlen(line) - 1 ] = '\0';
    part = strtok(line," ");
      i=0;
    while(part != NULL){
            tau[i++] = atoi(part);
          part = strtok(NULL," ");
    }

      deltaArray = (int ***)calloc(m,sizeof(int**));

      for(i=0;i<m;i++){
            deltaArray[i] = (int **)calloc(n,sizeof(int*));
            for(k=0;k<n;k++)
                  deltaArray[i][k] = (int*)calloc(n,sizeof(int));
            for(k=0;k<n;k++){
                  fgets(line, 80, exampleFile);
                  line[ strlen(line) - 1 ] = '\0';
                  part = strtok(line," ");
                  j = 0;
                  while(part != NULL){
```

ReducedNerode.c

```
                        deltaArray[i][k][j++] = atoi(part);
                        part = strtok(NULL," ");
                  }
            }
      }

   printf("Parsed numbers: n = %d, m = %d\n",n,m);
   for(i=0;i<n;i++)
         printf("%d ",sigma[i]);
   printf("\n");
   for(i=0;i<n;i++)
         printf("%d ",tau[i]);
   printf("\n");

   printf("\n****************** DELTA ***************\n");
   for(i=0;i<m;i++){
         printf("\n*************** MATRIX NO. %d
*****************\n",i);
         for(j=0;j<n;j++){
               for(k=0;k<n;k++)
                     printf("%d ",deltaArray[i][j][k]);
               printf("\n");
         }
   }

   fclose(exampleFile);
}

void print(tree t){
      int i,j;
      printf("\n************** TREE NODE: %lu ***********\n",t);
      printf("\n************** FIRST CHILD: %lu ***********\n",t-
>firstchild);
      printf("\n************** NEXT SIBILING: %lu ***********\n",t-
>nextsibling);
      printf("\n************** EQUAL TO: %lu ***********\n",t-
>equalWith);
      printf("LETTER: %d\n",t->letter);
      printf("LABEL: %d\n",t->label);
      printf("KAPA: %d\n",t->kapa);
      printf("***************** ETAU *************\n");
      for(i=0;i<m;i++){
            for(j=0;j<n;j++){
                  printf("%d ",t->etau[i][j]);
            }
            printf("\n");
      }
      printf("\n");
}

void computeReducedNerodAutomaton(){
```

ReducedNerode.c

```
      int i,j,k;
      tree treenode,newTreeNode,tmp;
      root = (tree) malloc(sizeof(struct node));

      //constructing the treee
      root->kapa = scalarProduct(sigma,tau,n);
      root->etau = (int **)calloc(m,sizeof(int*));
      for(i=0;i<m;i++){
            root->etau[i] = multiplication(sigma,deltaArray[i]);
      }
      root->letter = -1;
      root->equalWith = NULL;
      root->nextsibling = NULL;
      root->firstchild = NULL;
      root->label = 0;
      q = addToQueue(q,root);
      nOut = 1;
      while(q){
            treenode = deleteFromQueue(&q);
            for(i=0;i<m;i++){
                  newTreeNode = (tree)malloc(sizeof(struct node));
                  newTreeNode->equalWith = NULL;
                  newTreeNode->firstchild = NULL;
                  newTreeNode->nextsibling = NULL;
                  newTreeNode->letter = i;
                  newTreeNode->kapa = scalarProduct(treenode-
>etau[i],tau,n);
                  newTreeNode->etau = (int **)calloc(m,sizeof(int*));
                  for(k=0;k<m;k++){
                        newTreeNode->etau[k] = multiplication(treenode-
>etau[i],deltaArray[k]);
                  }
                  equalToTreeNode(newTreeNode);
                  addToTree(treenode,newTreeNode);
                  if(newTreeNode->equalWith == NULL){
                        nOut++;
                        newTreeNode->label = nOut-1;
                        q = addToQueue(q,newTreeNode);
                  } else {
                        newTreeNode->label = newTreeNode->equalWith-
>label;
                  }
            }
      }

      mOut = m;
      tauOut = (int *)calloc(nOut,sizeof(int));
      sigmaOut = (int *)calloc(nOut,sizeof(int));
      deltaArrayOut = (int ***)calloc(m,sizeof(int**));

      //constructing output delata array
```

ReducedNerode.c

```
	for(i=0;i<m;i++){
		deltaArrayOut[i] = (int **)calloc(nOut,sizeof(int*));
		for(k=0;k<nOut;k++)
			deltaArrayOut[i][k] = (int*)calloc(nOut,sizeof(int));
		for(k=0;k<nOut;k++)
			for(j=0;j<nOut;j++)
				deltaArrayOut[i][k][j] = 0;
	}

	//walk through the tree
	q = NULL;
	q = addToQueue(q,root);
	while(q){
		treenode = deleteFromQueue(&q);
		print(treenode);
		if(treenode->equalWith == NULL){
			tauOut[treenode->label] = treenode->kapa;
		}
		if(treenode->firstchild){
			q = addToQueue(q, treenode->firstchild);
			tmp = treenode->firstchild->nextsibling;
			//pupulating output delta array
			deltaArrayOut[treenode->firstchild->letter][treenode-
>label][treenode->firstchild->label] = 1;
			while(tmp){
				deltaArrayOut[tmp->letter][treenode->label][tmp-
>label] = 1;
				q = addToQueue(q, tmp);
				tmp = tmp->nextsibling;
			}
		}
	}

	//populating sigma array
	sigmaOut[0]=1;
	for(i=1;i<nOut;i++)
		sigmaOut[i] = 0;
}

void reverseAutomation(int *sigma, int *tau, int n, int ***deltas, int
m){
	int i,j,k, tmp;
	//exchanging sigma and tay arrays
	for(i=0;i<n;i++){
		tmp = sigma[i];
		sigma[i] = tau[i];
		tau[i] = tmp;
	}
	for(k=0;k<m;k++){
		for(i=0;i<n;i++){
			for(j=i+1;j<n;j++){
```

ReducedNerode.c

```
                        tmp = deltas[k][i][j];
                        deltas[k][i][j] = deltas[k][j][i];
                        deltas[k][j][i] = tmp;
                  }
            }
      }
}

void outputDataToInputData(){
      int i,j,k;

      free(sigma);
      free(tau);
      sigma = (int *)calloc(nOut,sizeof(int));
      tau = (int *)calloc(nOut,sizeof(int));
      for(i=0;i<nOut;i++){
            sigma[i] = sigmaOut[i];
            tau[i] = tauOut[i];
      }
      free(sigmaOut);
      free(tauOut);

      //free delta input array

      for(i=0;i<m;i++)
            for(j=0;j<n;j++)
                  free(deltaArray[i][j]);
      for(i=0;i<m;i++)
            free (deltaArray[i]);
      free(deltaArray);

      deltaArray = (int ***)calloc(mOut,sizeof(int**));
      for(i=0;i<mOut;i++){
            deltaArray[i] = (int **)calloc(nOut,sizeof(int*));
            for(k=0;k<nOut;k++)
                  deltaArray[i][k] = (int*)calloc(nOut,sizeof(int));
      }

      for(i=0;i<mOut;i++)
            for(j=0;j<nOut;j++)
                  for(k=0;k<nOut;k++)
                        deltaArray[i][j][k] = deltaArrayOut[i][j][k];

      //free delta output array

      for(i=0;i<mOut;i++)
            for(j=0;j<nOut;j++)
                  free(deltaArrayOut[i][j]);
      for(i=0;i<mOut;i++)
            free (deltaArrayOut[i]);
      free(deltaArrayOut);
```

ReducedNerode.c

```
      n = nOut;
      m = mOut;
}

void writeOutputFile(char *outputFileName){
      int i,j,k;
      FILE *g;
      g = fopen(outputFileName,"w");
      fprintf(g,"%d %d\n",nOut,mOut);
      //write sigma array
      for(i=0;i<nOut-1;i++)
            fprintf(g,"%d ",sigmaOut[i]);
      fprintf(g,"%d\n",sigmaOut[nOut-1]);
      //write tau array
      for(i=0;i<nOut-1;i++)
            fprintf(g,"%d ",tauOut[i]);
      fprintf(g,"%d\n",tauOut[nOut-1]);

      for(i=0;i<mOut;i++){
            for(j=0;j<nOut;j++){
                  for(k=0;k<nOut-1;k++){
                        fprintf(g,"%d ",deltaArrayOut[i][j][k]);
                  }
                  fprintf(g,"%d\n",deltaArrayOut[i][j][nOut-1]);
            }
      }
      fclose(g);
}

void main(){
      char* inputFile = "primer1.txt";
      char* outputFile;
      loadFromFile(inputFile);

      computeReducedNerodAutomaton();
      outputFile = (char *) calloc(100,sizeof(char));
      strcpy(outputFile,"result reduced nerode - ");
      strcat(outputFile,inputFile);
      //write result
      writeOutputFile(outputFile);
}
```

# Appendix B
# Algoritmi za determinizaciju težinskih i fazi automata

Determinizacija nedeterminističkih konačnih automata predstavlja jedan od najvažnijih problema u teoriji automata. To je proces transformacije nedeterminističkog automata u njemu jezički ekvivalentan deterministički automat. Determinizacija ima važnu ulogu u mnogim oblastima računarskih nauka kao što konstrukcija prevodilaca, procesiranje teksta, procesiranje prirodnih jezika, verifikacija i testiranje sistema i slično, ali i u oblastima izvan računarstva, kao što je molekularna biologija.

Postoji veliki broj algoritama za determinizaciju nedeterminističkih automata. Standardni algoritam za determinizaciju nedeterminističkih automata je, takozvana, podskup konstrukcija. Ovaj algoritam prevodi dati nedeterministički automat sa $n$ stanja u jezički ekvivalentan deterministički automat sa najvie $2^n$ stanja. Primenom podskup konstrukcije, u najgorem slučaju, dobijamo deterministički automat čiji je broj stanja eksponencijalno veći od broja stanja polaznog nedeterminističkog automata. Ipak, ovaj determinizacioni algoritam je poznat po svojim dobrim performansama u praksi. U skorije vreme, razvijeno je nekoliko algoritama baziranih na podkup konstrukciji, koji su memorijski efikasniji i koji kao rezultat daju automat sa manjim brojem stanja od onog dobijenog korišćenjem podskup konstrukcije. Jedan od najpoznatijih algoritama za determinizaciju nedeterminističkih automata je algoritam Brzozowskog. Ovaj algoritam proizvodi minimalni deterministički automat ekvivalentan originalnom automatu. I pored toga što je u najgorem slučaju kompleksnost ovog algoritma eksponencijalna, pokazalo se da se u mnogim praktičnim primerima on izvršava znatno brže nego što se, na osnovu njegove kompleksnosti očekuje.

Težinski i fazi automati su klasični nedeterministički automati kod kojih prelazi, inicijalna i završna stanja uzimaju vrednosti iz odredjenih struktura. Kod težinskih automata ove vrednosti, koje se najčešće se uzimaju iz poluprstena se nazivaju težine, dok se kod fazi automata one nazivaju istinitosne vrednosti i uzimaju se iz odredjenih uredjenih, (najčešće mrežno uredjenih) struktura. Ove vrednosti mogu da se koriste za modeliranje

verovatnoće uspešnog izvršenja prelaza ili količine energije potrebne za uspešno izvršenje prelaza. Pod determinizacijom teinskih (fazi) konačnih automata podrazumevamo postupak njihovog prevodjenja u ekvivalentan deterministički težinski (fazi) automat. Ovu vrstu determinizacije prvi su proučavali Belohlavek, za konačne fazi automate nad kompletnim mreama i Li i Pedrycz, za konačne fazi automate nad mreno uredjenim monoidima i dali su postupak za determinizaciju fazi automata koji predstavlja uoptenje podskup konstrukcije. Jo jedan algoritam, koji takodje predstavlja uoptenje podskup konstrukcije (Nerodov automat) dali su Ignjatović, Ćirić i Bogdanović. Nedavno, mnogi autori su se bavili proučavanjem težinskih automata kod kojih su težine uzete iz nekih opštijih struktura, nazvanih jaki bimonoidi, koji mogu biti definisani kao poluprsteni kod kojih ne mora da vai zakon distributivnosti. M. Droste, T. Stuber i H. Vogler su, nedavno počeli proučavanje konačnih teinskih automata nad proizvoljnim jakim bimonoidima i M. Droste, H. Vogler, M. Ćirić i J. Ignjatović su razvili metode i algoritme za determinizaciju ovih automata. Zbog odsustva distributivnosti, postoje tri različite definicije ponašanja težinskih automata nad jakim bimonoidima: run semantika, inicijalna algebarska semantika i tranziciona semantika. Nerodov automat je takodje konstruisan za težinski automat nad jakim bimonoidom i dokazano je da je on ekvivalentan polaznom težinskom automatu u odnosu na inicijalnu algebarsku semantiku.

Glavni zadatak predložene doktorske disertacije bio bi razvijanje novih metoda i algoritama za determinizaciju težinskih i fazi automata, od kojih neki predstavljaju metode za kanonizaciju tj. istovremeno vrše i determinizaciju i minimizaciju polaznih automata.

U prvoj glavi bili bi uvedeni osnovni pojmovi iz teorije mreža, sa posebnim osvrtom na reziduirane mreže, koje služe kao struktura istinitosnih vrednosti pri izučavanju fazi skupova. Dalje, bili bi uvedeni osnovni pojmovi teorije fazi skupova i fazi relacija, a zatim bi bili definisani osnovni pojmovi teorije poluprstena, pojam jakih bimonoida, vektora i matrica nad poluprstenima i formalnih stepenih redova.

U drugoj glavi bili bi predstavljeni glavni pojmovi i rezultati teorije težinskih i fazi automata i jezika. U skladu sa tim bile bi razmatrane tri vrste ponašanja težinskih automata nad jakim bimonoidima: ran sematika, inicijalna algebarska semantika i tranziciona semantika. Zatim bi bili posmatrani deterministički težinski, odnosno fazi automati. Dalje, bili bi razmatrani faktor težinski odnosno fazi automati u odnosu na relaciju kongruencije na skupu stanja datog automata.

U glavi tri bio bi dat algoritam za determinizaciju teinskih automata nad jakim bimonoidima koji generiše deterministički težinski automat ekvivalentan originalnom automatu. Bilo bi dokazano da je kardinalnost dobijenog automata uvek manja ili jednaka od kardinalnosti odgovarajućeg Nerodovog automata, a osim toga, pokazalo bi se da ovako generisan automat moe biti dobijen i redukcijom Nerodovog automata u odnosu na odgovarajuću kongruenciju. Zbog toga će on biti nazvan redukovani Nerodov automat. Ovde

bi bili dati potrebni i dovoljni uslovi pri kojima je redukovani Nerodov automat konačan.

U glavi četiri bio bi razvijen metod za determinizaciju težinskih automata nad komutativnim poluprstenima, koji predstavlja uopštenje algoritma Brzozowskog za determinizaciju klasičnih nedeterminističkih automata. Ovaj metod bi se zasnivao na sukcesivnoj primeni dve jednostavne operacije: reverzije i konstrukcije Nerodeovog automata. Kao rezultat ovaj algoritam bi davao minimalni deterministički težinski automat ekvivalentan polaznom automatu.

U petoj glavi panja bi bila posvećena metodama za determinizaciju fazi automatima nad kompletnim reziduiranim mreama, koje istovremeno vrše i redukciju automata. Tačnije, bila bi predstavljeni algoritmi koji simultano izvršavaju determinizaciju i redukciju polaznih fazi automata.

U šestoj glavi takodje bi bio razmatran metod za determinizaciju fazi automata nad kompletnim reziduiranim mreama, takozvana determinizacija pomoću stepena inkluzije. Ova determinizacija predstavlja istovremeno i determinizacioni i minimizacioni algoritam.

U nastavku navodimo glavne rezultate predložene doktorske disertacije.

## 1. Uvodni pojmovi i rezultati

*Jaki bimonoid* je struktura $(K,+,\cdot,0,1)$ koja zadovoljava sledeće: $(K,+,0)$ je monoid sa neutralnim elementom 0, $(K,\cdot,1)$ je monoid sa neutralnim elementom 1, operacija + je komutativna i 0 je multiplikativna nula, tj., $k\cdot 0 = 0 = 0\cdot k$ za svako $k \in K$. Za jaki bimonoid $K$ kažemo da je *desno distributivan*, ako zadovoljava $(a+b)\cdot c = a\cdot c + b\cdot c$, za svako $a,b,c \in K$, a kažemo da je $K$ levo distributivan, ako $a\cdot(b+c) = a\cdot b + a\cdot c$ za svako $a,b,c \in K$.

*Poluprsten* je struktura $(S,+,\cdot,0,1)$ takva da važi: $(S,+,0)$ je komutativan monoid, $(S,\cdot,1)$ je monoid, distributivni zakoni važe za sve $a,b,c \in S$ i $0\cdot a = a\cdot 0 = 0$ za svako $a \in S$. Drugim rečima, poluprsten je jaki bimonoid koji je desno i levo distributivan. Za poluprsten $S$ kažemo da je komutativan ako $a\cdot b = b\cdot a$ za svako $a,b \in S$. Poluprsten $S$ je lokalno konačan ako je svaki njegov konačno generisan podpoluprsten konačan. Na osnovu ovoga imamo da je poluprsten $(S,+,\cdot,0,1)$ lokalno konačan akko su oba monoida $(S,+,0)$ i $(S,\cdot,1)$ lokalno konačna.

*Vektor* nad nepraznim skupom $A$ sa vrednostima u $S$ je preslikavanje $\nu : A \to S$. *Matrica* nad $A$ i $B$ sa vrenostima u $S$ je takodje preslikavanje $\mu : A\times B \to S$, tj. , to je proizvoljan vektor nad $A\times B$ sa vrednostima u $S$.

Za neprazne skupove $A$, $B$ and $C$ i matrice $\mu_1 : A\times B \to S$ and $\mu_2 : B\times C \to S$, *proizvod* $\varphi\cdot\psi$ je matrica nad $A$ i $C$ definisana sa

$$(\mu_1\cdot\mu_2)(a,c) = \sum_{b\in B} \mu_1(a,b)\cdot\mu_2(b,c), \tag{B.1}$$

za sve $a \in A$ i $c \in C$. Dalje, ako su $\nu_1 : A \to S$, $\mu : A\times B \to S$ i $\nu_2 : B \to S$, proizvodi $\nu_1\cdot\mu$ i $\mu\cdot\nu_2$ su vektori nad $B$ i $A$, respektivno, definisani sa

$$(\nu_1 \cdot \mu)(b) = \sum_{a \in A} \nu_1(a) \cdot \mu(a,b), \quad (\mu \cdot \nu_2)(a) = \sum_{b \in B} \mu(a,b) \cdot \nu_2(b), \tag{B.2}$$

za sve $a \in A$ i $b \in B$. Proizvod dva vektora $\nu_1$ i $\nu_2$ nad $A$ je element skupa $S$ dat sa:

$$\nu_1 \cdot \nu_2 = \bigvee_{a \in A} \nu_1(a) \otimes \nu_2(a). \tag{B.3}$$

*Formalni stepeni red* nad $X$ i $K$ (skrćeno *red*), je proizvoljno preslikavanje $\varphi : X^* \to K$. Umesto $\varphi(u)$ pišemo $(\varphi, u)$ za svako $u \in X^*$.Skup svih redova nad $X$ i $K$ se označava sa $K\langle\langle X^* \rangle\rangle$.

*Reziduirana mreža* je algebra $\mathscr{L} = (L, \wedge, \vee, \otimes, \to, 0, 1)$ takva da

(L1) $(L, \wedge, \vee, 0, 1)$ je mreža sa najmanjim elementom 0 i najvećim 1,
(L2) $(L, \otimes, 1)$ je komutativni monoid sa jedinicom 1,
(L3) $\otimes$ i $\to$ formiraju *adjungovani par* , tj. zadovoljavaju *svojstvo adjungcije*: za sve $x, y, z \in L$,

$$x \otimes y \leqslant z \;\Leftrightarrow\; x \leqslant y \to z. \tag{B.4}$$

Ako je osim toga $(L, \wedge, \vee, 0, 1)$ kompletna mreža, tada $\mathscr{L}$ nazivamo *kompletna reziduirana mreža* .

Operacija $\otimes$ (zvana *multiplikacija*) i $\to$ (zvana *reziduum*) namenjene su za modeliranje konjunkcije i implikacije odgovarajućih logičih računa, i supremum ($\bigvee$) i infimum ($\bigwedge$) namenjeni su za modeliranje ekstenzionalnog i univerzalnog kvantifikatora, respektivno. Operacija $\leftrightarrow$ definisana sa

$$x \leftrightarrow y = (x \to y) \wedge (y \to x), \tag{B.5}$$

naziva se *bireziduum* (ili *biimplikacija*), i koristi se za modeliranje ekvivalencije istinitosnih vrednosti.

*Fazi podskup* skupa $A$ *nad* $\mathscr{L}$, ili samo *fazi podskup* od $A$, je svako preslikavanje iz $A$ u $L$. *Jednakost* fazi poskupova $f$ i $g$ definiše se obično kao jednakost preslikavanja, tj. $f = g$ ako i samo ako $f(x) = g(x)$, za svako $x \in A$. *Inkluzija* $f \leqslant g$ takodje se definiše kao jednakost preslikavanja: $f \leqslant g$ ako i samo ako $f(x) \leqslant g(x)$, za svako $x \in A$. Zajedno sa ovim parcijalnim uredjenjem skup $L^A$ svih fazi podskupova od $A$ formira kompletnu reziduiranu mrežu, u kojoj su presek $\bigwedge_{i \in I} f_i$ i unija $\bigvee_{i \in I} f_i$ proizvoljne familije $\{f_i\}_{i \in I}$ fazi podskupova od $A$ preslikavanja iz $A$ u $L$ definisana sa

$$\left(\bigwedge_{i \in I} f_i\right)(x) = \bigwedge_{i \in I} f_i(x), \qquad \left(\bigvee_{i \in I} f_i\right)(x) = \bigvee_{i \in I} f_i(x),$$

i *proizvod* $f \otimes g$ je fazi poskup definisan sa: $f \otimes g(x) = f(x) \otimes g(x)$, za svako $x \in A$.

*Fazi relacija* izmedju skupova $A$ i $B$ je svako preslikavanje iz $A \times B$ u $L$, tj. svaki fazi podskup od $A \times B$, i jednakost, inkluzija (uredjenje), unija i presek fazi relacija definisani su kao u slučaju fazi skupova. Skup svih fazi relacija

izmedju $A$ i $B$ označićemo sa $L^{A\times B}$. Specijalno, fazi relacija na skupu $A$ je svaka funkcija iz $A\times A$ u $L$, tj fazi poskup od $A\times A$. Skup svih fazi relacija na $A$ biće označen sa $L^{A\times A}$. *Inverz* fazi relacija $\alpha\in L^{A\times B}$ je fazi relacija $\alpha^{-1}\in L^{B\times A}$ definisana sa $\alpha^{-1}(b,a)=\alpha(a,b)$, za svako $a\in A$ i $b\in B$. Krisp relacija je fazi relacija koja uzima vrednosti samo u skupu $\{0,1\}$, i ako je $\alpha$ krisp relacija iz $A$ u $B$, tada izrazi "$\alpha(a,b)=1$" i "$(a,b)\in\alpha$" imaju isto značenje.

Za neprazne skupove $A$, $B$ i $C$, i fazi relaciju $\alpha\in L^{A\times B}$ i $\beta\in L^{B\times C}$, njihova *kompozicija* $\alpha\circ\beta\in L^{A\times C}$ je fazi relacija definisana sa

$$(\alpha\circ\beta)(a,c)=\bigvee_{b\in B}\alpha(a,b)\otimes\beta(b,c), \tag{B.6}$$

za sve $a\in A$ i $c\in C$. Za $f\in L^A$, $\alpha\in L^{A\times B}$ i $g\in L^B$, kompozicije $f\circ\alpha\in L^B$ i $\alpha\circ g\in L^A$ su fazi skupovi definisani

$$(f\circ\alpha)(b)=\bigvee_{a\in A}f(a)\otimes\alpha(a,b),\qquad(\alpha\circ g)(a)=\bigvee_{b\in B}\alpha(a,b)\otimes g(b), \tag{B.7}$$

za svako $a\in A$ i $b\in B$. Konačno, kompozicija fazi skupova $f,g\in L^A$ je element $f\circ g\in L$ (skalar) definisan sa

$$f\circ g=\bigvee_{a\in A}f(a)\otimes g(a). \tag{B.8}$$

Fazi relacija $\varphi$ na skupu $A$ naziva se *refleksivna*, ako $\varphi(a,a)=1$, *simetrična*, ako $\varphi(a,b)=\varphi(b,a)$, i *transitivna*, ako $\varphi(a,b)\otimes\varphi(b,c)\leqslant\varphi(a,c)$, za sve $a,b,c\in A$. Refleksivna i tranzitivnafazi relacija naziva se *fazi kvazi uredjenje*. Simetrično fazi kvazi uredjenje je *fazi ekivalencija*. Za kvazi uredjenje $\varphi$ na $A$ i element $a\in A$, *$\varphi$-afterset* od $a$ je fazi skup $a\varphi\in L^A$ definisan sa $a\varphi(b)=\varphi(a,b)$, i *$\varphi$-foreset* od $a$ je fazi skup $\varphi a\in L^A$ definisan sa $\varphi a(b)=\varphi(b,a)$, za sve $b\in A$. Ako je $\varphi$ fazi ekvivalencija, tada se *$\varphi$-afterset* od $a$ poklapa sa *$\varphi$-foresetom* od $a$, i naziva se *fazi klasa ekvivalencije* od $a$.

## 2. Težinski i fazi automati

Neka je $K$ jaki bimonoid i $X$ (konačan) alfabet. *Težinski automat* nad $X$ i $K$ je četvorka $\mathscr{A}=(A,\delta,\sigma,\tau)$, pri čemu je $A$ konačan neprazan skup, *skup stanja*, $\delta:A\times X\times A\to K$ je *težinska funkcija prelaza*, $\sigma:A\to K$ *inicijalni vektor* i $\tau:A\to K$ *završni vektor*. Za svako $x\in X$ definišemo *težinsku matricu prelaza* (odnosno *težinsku relaciju prelaza*) $\delta_x:A\times A\to K$ sa $\delta_x(a,b)=\delta(a,x,b)$, za sve $a,b\in A$. Ako je $\mathscr{A}$ konačan težinski automat, onda $|\mathscr{A}|=|A|$, tj., $|\mathscr{A}|$ označava broj stanja automata $\mathscr{A}$.

Zbog odsustva distributivnosti u $K$, ponašanje težinskog automata može biti definisano na više različitih načina. Konkretno, razlikujemo *inicijalnu algebarsku semantiku*,*ran semantiku* i *tranzicionu semantiku*.

*Inicijalna algebarska semantika*: Za svako $u \in X^*$ definišemo vektor $\sigma_u : A \to K$ induktivno, na sledeći način: $\sigma_\varepsilon = \sigma$, i za sve $u \in X^*$ i $x \in X$ stavljamo $\sigma_{ux} = \sigma_u \cdot \delta_x$. Očigledno, ako je $u = x_1 x_2 \cdots x_n$, onda

$$\sigma_u = (\ldots((\sigma \cdot \delta_{x_1}) \cdot \delta_{x_2}) \cdot \ldots) \cdot \delta_{x_n}. \tag{B.9}$$

Red $[\![\mathscr{A}]\!]_i$ u $K\langle\langle X^*\rangle\rangle$ definisan sa

$$([\![\mathscr{A}]\!]_i, u) = \sigma_u \cdot \tau = ((\ldots((\sigma \cdot \delta_{x_1}) \cdot \delta_{x_2}) \cdot \ldots) \cdot \delta_{x_n}) \cdot \tau\,. \tag{B.10}$$

za svako $u = x_1 x_2 \cdots x_n \in X^*$, se naziva *i-ponašanje* automata $\mathscr{A}$.

*Tranziciona semantika*: Za svako $u \in X^*$ definišemo vektor $\delta_u : A \times A \to K$ induktivno, na sledeći način: $\delta_\varepsilon$ je jedinična matrica, i za svako $u \in X^*$ i $x \in X$ stavljamo $\delta_{ux} = \delta_u \cdot \delta_x$. Drugim rečima, ako je $u = x_1 x_2 \cdots x_n \in X^*$, onda

$$\delta_u = (\ldots((\delta_{x_1} \cdot \delta_{x_2}) \cdot \delta_{x_3}) \cdot \ldots) \cdot \delta_{x_n}. \tag{B.11}$$

Red $[\![\mathscr{A}]\!]_t$ u $K\langle\langle X^*\rangle\rangle$ definisan sa

$$([\![\mathscr{A}]\!]_t, u) = (\sigma \cdot \delta_u) \cdot \tau \tag{B.12}$$

za svako $u \in X^*$, se naziva *t-ponašanje* automata $\mathscr{A}$.

*Ran semantika*: Red $[\![\mathscr{A}]\!]_r$ u $K\langle\langle X^*\rangle\rangle$ definisan sa

$$([\![\mathscr{A}]\!]_r, u) = \sum_{(a_0,a_1,\ldots,a_n)\in A^{n+1}} \sigma(a_0) \cdot \delta_{x_1}(a_0,a_1) \cdot \delta_{x_2}(a_1,a_2) \cdot \ldots \cdot \delta_{x_n}(a_{n-1},a_n) \cdot \tau(a_n)\,. \tag{B.13}$$

za svako $u = x_1 x_2 \cdots x_n \in X^*$, se naziva *r-ponašanje* automata $\mathscr{A}$.

Neka je $s \in \{i,t,r\}$. Red $\varphi \in K\langle\langle X^*\rangle\rangle$ se naziva *s-raspoznatljiv* ako postoji konačan težinski automat $\mathscr{A}$ nad $X$ i $K$ tako da $[\![\mathscr{A}]\!]_s = \varphi$. Kažemo da su dva težinska automata $\mathscr{A}$ and $\mathscr{A}'$ nad $X$ i $K$ *s-ekvivalentni* ako $[\![\mathscr{A}]\!]_s = [\![\mathscr{A}']\!]_s$.

U nastavku, $\mathscr{L}$ će biti kompletna reziduirana mreža i $X$ će biti (konačan) alfabet. *Fazi automat nad* $\mathscr{L}$ i $X$ , ili samo *fazi automat*, je uredjena četvorka $\mathscr{A} = (A, \delta, \sigma, \tau)$, gde je: $A$ neprazan skup (*skup stanja automata*) , $\delta : A \times X \times A \to L$ je fazi poskup od $A \times X \times A$ (*fazi tranziciona funkcija*), $\sigma : A \to L$ je fazi podskup od $A$ (*fazi skup inicijalnih stanja*), a $\tau : A \to L$ je fazi podskup od $A$ (*fazi skup završnih stanja*). Vrednost $\delta(a,x,b)$ možemo interpretirati kao stepen do koga ulazno slovo $x \in X$ uzrokuje prelaz iz stanja $a \in A$ u stanje $b \in A$, dok vrednosti $\sigma(a)$ i $\tau(a)$ mozemo interpretirati kao stepen do kog $a$ pripada skupu inicijalnih odnosno završnih stanja. Iz metodoloških razloga ćemo dozvoliti da skup stanja $A$ bude beskonačan. Fazi automa čiji je skup stanja konačan naziva se *konačan fazi automat*.

*Reverzni fazi automat* automata $\mathscr{A} = (A, \delta, \sigma, \tau)$ je definisan kao fazi automat $\overline{\mathscr{A}} = (A, \bar{\delta}, \bar{\sigma}, \bar{\tau})$ čija je fazi tranziciona funkcija prelaza data sa:

$$\bar{\delta}(a_1, x, a_2) = \delta(a_2, x, a_1) \quad \text{za sve } a_1, a_2 \in A, \quad x \in X,$$

a fazi inicijalna i završna stanja su $\bar{\sigma} = \tau$ i $\bar{\tau} = \sigma$.

Fazi automat $\mathscr{A} = (A, \delta^A, \sigma^A, \tau^A)$ i $\mathscr{A}' = (A', \delta^{A'}, \sigma^{A'}, \tau^{A'})$ su *izomorfni* ako postoji bijekcija $\phi : A \to A'$ takva da $\delta_x^A(a,b) = \delta_x^{A'}(\phi(a), \phi(b))$, za sve $a, b \in A$ i $x \in X$, i takodje, $\sigma^A(a) = \sigma^{A'}(\phi(a))$ i $\tau^A(a) = \tau^{A'}(\phi(a))$, za svako $a \in A$.

*Fazi jezik* u $X^*$ nad $\mathscr{L}$, ili samo *fazi jezik*, je svaki podskup skupa $X^*$, tj. svaka funkcija iz $X^*$ u $L$. *Fazi jezik raspoznat fazi automatom* $\mathscr{A} = (A, \delta, \sigma, \tau)$, u oznaci $[\![\mathscr{A}]\!]$, je fazi jezik $\mathscr{F}(X^*)$ definisan sa

$$\begin{aligned} [\![\mathscr{A}]\!](e) &= \sigma^A \circ \tau^A, \\ [\![\mathscr{A}]\!](u) &= \sigma^A \circ \delta_{x_1}^A \circ \delta_{x_2}^A \circ \cdots \circ \delta_{x_n}^A \circ \tau^A, \end{aligned} \tag{B.14}$$

za sve $u = x_1 x_2 \dots x_n \in X^+$, gde je $x_1, x_2, \dots, x_n \in X$.

Fazi automati $\mathscr{A}$ i $\mathscr{B}$ su *jezički ekvivalentni*, ili samo *ekvivalentni*, ako je $[\![\mathscr{A}]\!] = [\![\mathscr{B}]\!]$.

Za dati fazi jezik $\varphi : X^* \to L$ i $v \in X^*$, definišemo $v^{-1}\varphi : X^* \to L$ i $\varphi v^{-1} : X^* \to L$ sa $v^{-1}\varphi(u) = \varphi(vu)$ i $\varphi v^{-1}(u) = \varphi(uv)$ za $u \in X^*$. Fazi jezik $v^{-1}\varphi$ naziva se *levi derivat*, a fazi jezik $\varphi v^{-1}$ *desni derivat* od $\varphi$ u odnosu na $v$.

Kardinalnost fazi automata $\mathscr{A} = (A, \delta^A, \sigma^A, \tau^A)$, u oznaci $|\mathscr{A}|$, se definiše kao kardinalnost skupa stanja automata $\mathscr{A}$. Fazi automat $\mathscr{A}$ je *minimalni fazi automat* jezika $f \in \mathscr{F}(X^*)$ ako raspoznaje jezik $f$ i $|\mathscr{A}| < |\mathscr{A}'|$, za svaki automat $\mathscr{A}'$ koji raspoznaje $f$. Minimalni fazi automat koji raspoznaje dati fazi jezik $f$ ne mora obavezno biti jedinstven do na izomorfizam. Ovo takodje važi i u slučaju nedeterminističkih automata.

Neka je $\mathscr{A} = (A, \delta, \sigma, \tau)$ fazi automat nad $X$ i $\mathscr{L}$, respektivno težinski automat nad $X$ i $K$. Fazi (težinska) funkcija prelaza $\delta$ je *deterministička* ako za svaki $x \in X$ i svako $a \in A$ postoje $a' \in A$ takvi da $\delta_x(a, a') = 1$, i $\delta_x(a, b) = 0$, za svako $b \in A \setminus \{a'\}$. Za fazi skup inicijalnih stanja (težinska inicijalna funkcija) $\sigma$ kažemo da je *deterministička* ako postoji $a_0 \in A$ takvo da $\sigma(a_0) = 1$ i $\sigma(a) = 0$ za svako $a \in A \setminus \{a_0\}$. Ako su oba $\sigma$ i $\delta$ determinističke, tada sa $\mathscr{A}$ naziva *deterministički* fazi (težinski) automat (kraće: *dfa*, respektivno *dta*).

## 3. Redukovani Nerodov automat

U ovoj sekciji, ukoliko nije naglašeno drugačije, $K$ će biti jak bimonoid, i alfabet će biti $X = \{x_1, x_2, ..., x_n\}$, za neko $n \in \mathbb{N}$.

Neka je $\mathscr{A} = (A, \delta, \sigma, \tau)$ konačan težinski automat nad $X$ i $K$. Postavimo $A_N = \{\sigma_u \mid u \in X^*\}$, i definišimo $\delta_N : A_N \times X \to A_N$ i $\tau_N : A_N \to K$ sa

$$\delta_N(\sigma_u, x) = \sigma_{ux} \quad \text{i} \quad \tau_N(\sigma_u) = \sigma_u \cdot \tau, \tag{B.15}$$

za svako $u \in X^*$ i $x \in X$. Tada je $\mathscr{A}_N = (A_N, \delta_N, \sigma_\varepsilon, \tau_N)$ krisp-deterministički težinski automat nad $X$ i $K$, koji se naziva *Nerodov automat* od automata $\mathscr{A}$, i $\mathscr{A}_N$ je $i$-ekvivalentan sa $\mathscr{A}$, tj. $[\![\mathscr{A}_N]\!] = [\![\mathscr{A}]\!]_i$.

Neka je $\mathscr{A} = (A, \delta, \sigma, \tau)$ težinski automat nad $X$ i $K$. Za svaku reč $u \in X^*$ i svako slovo $x \in X$ definišimo vektor $\eta_u^x : A \to K$ sa

$$\eta_u^x = \sigma_{ux}, \tag{B.16}$$

tj., $\eta_u^x = \sigma_u \cdot \delta_x$. Osim toga, za svaku reč $u \in X^*$ definišemo skalar $\kappa_u \in K$ sa

$$\kappa_u = \sigma_u \cdot \tau. \tag{B.17}$$

Dalje, za svaku reč $u \in X^*$ definišemo *težinsku tranzicionu* $(n+1)$*-torku* $\theta_u$ sa

$$\theta_u = (\eta_u^{x_1}, \ldots, \eta_u^{x_n}, \kappa_u). \tag{B.18}$$

Naredna lema pokazuje da težinska tranziciona $(n+1)$-torka $\theta_{ux}$ može biti izračunata iz $\theta_u$, za proizvoljno $u \in X^*$ i $x \in X$.

**Lema 2.1.** *Neka je $\mathscr{A} = (A, \delta, \sigma, \tau)$ konačan težinski automat nad $X$ i $K$. Tada za sve $u \in X^*$ i $x \in X$ važi sledeće:*

$$\theta_{ux} = (\eta_u^x \cdot \delta_{x_1}, \ldots, \eta_u^x \cdot \delta_{x_n}, \eta_u^x \cdot \tau). \tag{B.19}$$

Sada, postavimo $A_R = \{\theta_u \mid u \in X^*\}$, i definišimo $\delta_R : A_R \times X \to A_R$ i $\tau_R : A_R \to K$ sa

$$\delta_R(\theta_u, x) = \theta_{ux} \quad \text{i} \quad \tau_R(\theta_u) = \kappa_u, \tag{B.20}$$

za svako $u \in X^*$ i $x \in X$.

Tada važi sledeće.

**Teorema 2.1.** *Neka je $\mathscr{A} = (A, \delta, \sigma, \tau)$ konačan težinski automat nad $X$ i $K$. Tada je $\mathscr{A}_R = (A_R, \delta_R, \theta_\varepsilon, \tau_R)$ deterministički težinski automat nad $X$ i $K$, i $\mathscr{A}_R$ je i-ekvivalentan sa $\mathscr{A}$, tj. $[\![\mathscr{A}_R]\!] = [\![\mathscr{A}]\!]_i$.*

Primetimo da za svaku reč $u \in X^*$ težinska tranziciona $(n+1)$-torka $\theta_u = (\eta_u^{x_1}, \ldots, \eta_u^{x_n}, \kappa_u)$ može biti posmatrana kao uredjeni par čija je prva komponenta $n$-torka $(\eta_u^{x_1}, \ldots, \eta_u^{x_n})$, a druga komponenta je $\kappa_u$. S druge strane, $n$-torka $(\eta_u^{x_1}, \ldots, \eta_u^{x_n})$ može biti identifikovana kao $X \times A$-matrica $\eta_u$ nad $K$ definisana sa

$$\eta_u(x, a) = \eta_u^x(a), \tag{B.21}$$

za sve $x \in X$ i $a \in A$, jer za proizvoljno $u, v \in X^*$ važi

$$\begin{aligned} \eta_u = \eta_v &\Leftrightarrow (\forall x \in X)(\forall a \in A)\ \eta_u(x,a) = \eta_v(x,a) \Leftrightarrow \\ &\Leftrightarrow (\forall x \in X)(\forall a \in A)\ \eta_u^x(a) = \eta_v^x(a) \Leftrightarrow \\ &\Leftrightarrow (\forall x \in X)\ \eta_u^x = \eta_v^x \Leftrightarrow (\eta_u^{x_1}, \ldots, \eta_u^{x_n}) = (\eta_v^{x_1}, \ldots, \eta_v^{x_n}). \end{aligned}$$

Odatle, $\theta_u$ može biti razmatran kao par $(\eta_u, \kappa_u)$.

Na osnovu (B.16) i (B.21), za sve $u \in X^*$, $x \in X$ i $a \in A$ imamo

$$\eta_u(x, a) = \sigma_{ux}(a) = (\sigma_u \cdot \delta_x)(a) = \sum_{b \in A} \sigma_u(b) \cdot \delta_x(b, a). \tag{B.22}$$

Specijalno, u slučaju kada je $\mathscr{A}$ nedeteministički (Bulov) automat, $\eta_u$ možemo prirodno interpretirati kao: $\eta_u$ je skup svih izlaznih tranzicija iz skupa stanja $\sigma_u$. U ovom slučaju, za razliku od podskup konstrukcije, gde se radi sa skupom stanja, konstrukcija $\mathscr{A}_R$ radi sa skupom tranzicija.

**Teorema 2.2.** *Neka je $\mathscr{A} = (A, \delta, \sigma, \tau)$ konačan težinski automat nad $X$ i $K$. Tada je $|\mathscr{A}_R| \leqslant |\mathscr{A}_N|$.*

Automat $\mathscr{A}_R$ može biti dobijen redukcijom Nerodovog automata $\mathscr{A}_N$ pomoću odgovarajuće kongruencije. Zbog toga ćemo $\mathscr{A}_R$ zvati *redukovani Nerodov automat* od $\mathscr{A}$.

Neka je $\mathscr{A} = (A, \delta, \sigma, \tau)$ konačan težinski automat nad $X$ i $K$. Za svaki $\tau' : A \to K$, težinski konačan automat $(A, \delta, \sigma, \tau')$ se naziva *završna varijanta* od $\mathscr{A}$, i sa $\mathrm{Rec}_i(\mathscr{A}, \sigma)$ označimo familiju svih redova u $K\langle\langle X^* \rangle\rangle$ koji su $i$-ponašanja neke završne varijante od $\mathscr{A}$. Za svaki $a \in A$, red $[\![\mathscr{A}]\!]^{(\sigma,a)}$ u $K\langle\langle X^* \rangle\rangle$ je definisan sa

$$([\![\mathscr{A}]\!]^{(\sigma,a)}, u) = \sigma_u(a),$$

za svako $u \in X^*$.

Dalje, za svaki konačan težinski automat $\mathscr{A} = (A, \delta, \sigma, \tau)$ nad $X$ i $K$, i svaki $x \in X$ i $a \in A$, definišimo red $[\![\mathscr{A}]\!]^{(\eta,x,a)}$ u $K\langle\langle X^* \rangle\rangle$ sa

$$([\![\mathscr{A}]\!]^{(\eta,x,a)}, u) = \eta_u^x(a)$$

za svako $u \in X^*$. Lako je pokazati da je $[\![\mathscr{A}]\!]^{(\eta,x,a)} = [\![\mathscr{A}_R^{(x,a)}]\!]$, gde je $\mathscr{A}_R^{(x,a)} = (A_R, \delta_R, \theta_\varepsilon, \tau_R^{(x,a)})$ završna varijanta od $\mathscr{A}_R$ sa konačnim težinskim vektorom $\tau_R^{(x,a)}$ definisanim sa $\tau_R^{(x,a)}(\theta_u) = \eta_u^x(a)$, za svako $u \in X^*$. Takodje, za svako $a \in A$, $x \in X$ i $u \in X^*$ imamo da

$$([\![\mathscr{A}]\!]^{(\eta,x,a)}, u) = \eta_u^x(a) = \sigma_{ux}(a) = ([\![\mathscr{A}]\!]^{(\sigma,a)}, ux),$$

i stoga, red $[\![\mathscr{A}]\!]^{(\eta,x,a)}$ je desni derivat reda $[\![\mathscr{A}]\!]^{(\sigma,a)}$ u odnosu na $x$.

Sledeća teorema daje potrebne i dovoljne uslove pod kojima je redukovani Nerodov automat konačan.

**Teorema 2.3.** *Neka je $\mathscr{A} = (A, \delta, \sigma, \tau)$ konačan težinski automat nad $X$ i $K$. Sledeći uslovi su ekvivalentni:*

(1) *Automat $\mathscr{A}_R$ je konačan;*
(2) *Nerodov automat $\mathscr{A}_N$ je konačan;*
(3) *Desna kongruencija $R_{\mathscr{A}}$, definisana sa $(u, v) \in R_{\mathscr{A}} \Leftrightarrow \theta_u = \theta_v$, $u, v \in X^*$ ima konačan indeks;*
(4) *Redovi $[\![\mathscr{A}]\!]^{(\eta,x,a)}$, za sve $x \in X$ i $a \in A$, i $[\![\mathscr{A}]\!]_i$ imaju konačne slike;*
(5) *Redovi $\varphi x^{-1}$, za sve $\varphi \in \mathrm{Rec}_i(\mathscr{A}, \sigma)$ i $x \in X$, i $[\![\mathscr{A}]\!]_i$ su dtka-raspoznatljivi.*

Sada, dajemo algoritam za računanje redukovanog Nerodovog automata.

**Algoritam 2.1. (*Konstrukcija automata* $\mathscr{A}_R$)** Ulaz algoritma je konačan težinski automat $\mathscr{A} = (A, \delta, \sigma, \tau)$ nad $X = \{x_1, ..., x_n\}$ i $K$, i izlaz je deterministički težinski automat $\mathscr{A}_R = (A_R, \delta_R, \theta_\varepsilon, \tau_R)$ koji je $i$-ekvivalentan sa $\mathscr{A}$.

Procedura se sastoji u konstrukciji *grafa (stabla) prelaza* od $\mathscr{A}_R$ direktno iz grafa prelaza od $\mathscr{A}$. Stablo se konstruiše induktivno na sledeći način:

(A1) Koren stabla je $\theta_\varepsilon = (\sigma \circ \delta_{x_1}, ..., \sigma \circ \delta_{x_n}, \sigma \circ \tau)$, i stavljamo $T_0 = \{\theta_\varepsilon\}$.

(A2) Nakon $i$-tog koraka drvo $T_i$ je konstruisano, i čvorovi $T_i$ su označeni sa 'zatvoren' ili 'ne-zatvoren'. Značenje ova dva pojma biće objašnjeno u nastavku.

(A3) U sledećem koraku konstruišemo stablo $T_{i+1}$ obogaćivanjem stablo $T_i$ na sledeći način: za svaki ne zatvoren list $\theta_u = (\eta_u^{x_1}, ..., \eta_u^{x_n}, \kappa_u)$ koji se javlja u $T_i$, gde je $u \in X^*$, i za svako $x \in X$ dodajemo čvor $\theta_{ux} = (\eta_u^x \cdot \delta_{x_1}, ..., \eta_u^x \cdot \delta_{x_n}, \eta_u^x \cdot \tau)$ i granu iz $\theta_u$ u $\theta_{ux}$ označenu sa $x$. Ako je, u nastavku, $\theta_{ux}$ stanje koje je već konstruisano, tada markiramo $\theta_{ux}$ kao *zatvoreno*. Procedura se završava kada su svi listovi markirani kao zatvoreni.

(A4) Simultano, za svaki ne zatvoren list $\theta_u$ koji se pojavljuje u $T_i$, gde je $u \in X^*$, računamo vrednost $\tau_R(\theta_u)$ pomoću formule (B.20).

(A5) Kada je stablo prelaza $\mathscr{A}_R$ konstruisano, brišemo sve znake zatvorenja i lepimo listove za unutrašnje čvorove sa istom oznakom. Dijagram koji se dobija je graf prelaza od $\mathscr{A}_R$.

Ako je poluprsten $K$ slabo lokalno konačan, tada se algoritam završava u konačnom broju koraka, za svaki težinski konačan automat nad $K$, i rezultat je deterministički konačan težinski automat.

S druge strane, ako $K$ nije lokalno konačan, tada se algoritam završava u konačnom broju koraka pod uslovima odredjenim Teoremom 2.3..

## 4. Determinizacija tipa Brzozowskog

Neka je $\mathscr{A} = (A, \delta, \sigma, \tau)$ fazi automat nad $X$ i $\mathscr{L}$. *Reverzni Nerodov automat* od $\mathscr{A}$ je Nerodov automat reverznog fazi automata $\overline{\mathscr{A}}$ od $\mathscr{A}$. Jednostavnosti radi, označićemo reverzni Nerodov automat od $\mathscr{A}$ sa $\mathscr{A}_{\overline{N}}$ (umesto sa $(\overline{\mathscr{A}})_N$). Primetimo da je $\mathscr{A}_{\overline{N}} = (A_{\overline{N}}, \delta_{\overline{N}}, \tau_\varepsilon, \tau_{\overline{N}})$, gde je $A_{\overline{N}} = \{\tau_u \mid u \in X^*\}$, i funkcija $\delta_{\overline{N}} : A_{\overline{N}} \times X \to A_{\overline{N}}$ i $\tau_{\overline{N}} : A_{\overline{N}} \to L$ su date sa

$$\delta_{\overline{N}}(\tau_u, x) = \tau_{xu}, \qquad \tau_{\overline{N}}(\tau_u) = \tau_u \circ \sigma, \tag{B.23}$$

za sve $u \in X^*$ i $x \in X$.

Za svako stanje $a \in A$, *desni fazi jezik povezan sa stanjem* $a$ je fazi jezik $\tau_a \in L^{X^*}$ definisan sa

$$\tau_a(u) = \bigvee_{b \in A} \delta^*(a, u, b) \otimes \tau(b),$$

za svako $u \in X^*$. Drugim rečima, $\tau_a$ je fazi jezik raspoznat fazi automatom $\mathscr{A}' = (A, \delta, a, \tau)$ dobijen iz $\mathscr{A}$ zamenom $\sigma$ sa jednim krisp inicijalnim stanjem $a$. *Levi fazi jezik povezan sa stanjem* $a$ je fazi jezik $\sigma_a \in L^{X^*}$ dat sa

$$\sigma_a(u) = \bigvee_{b\in A} \sigma(b) \otimes \delta^*(b,u,a),$$

za svako $u \in X^*$, tj. fazi jezik raspoznat fazi automatom $\mathscr{A}' = (A,\delta,\sigma,\{a\})$ dobijen iz $\mathscr{A}$ zamenom $\tau$ sa determinističkim završnim stanjem $a$.

**Lema 2.2.** *Desni fazi jezik povezan sa stanjem a fazi automata $\mathscr{A}$ je jednak reverzu levog fazi jezika povezanog sa stanjem a u reverznom fazi automatu $\overline{\mathscr{A}}$.*

Za deterministički fazi automat $\mathscr{A} = (A,\delta,a_0,\tau)$, desni fazi jezik povezan sa stanjem $a$ u $\mathscr{A}$ je dat sa

$$\tau_a(u) = \tau(\delta^*(a,u)), \tag{B.24}$$

za sve $u \in X^*$, i specijalno, $\tau_{a_0} = [\![\mathscr{A}]\!]$, tj. desni fazi jezik povezan sa inicijalnim stanjem $a_0$ je fazi jezik respoznat sa $\mathscr{A}$. Lako se može pokazati da važi i sledeće:

**Lema 2.3.** *Neka je $\mathscr{A} = (A,\delta,a_0,\tau)$ deterministički fazi automat. Tada*

$$\tau_{\delta^*(a,u)} = u^{-1}\tau_a, \tag{B.25}$$

*za sve $a \in A$ i $u \in X^*$.*

Ako je $\mathscr{A} = (A,\delta,a_0,\tau)$ deterministički fazi automat, definišemo drugi deterministički fazi automat $\mathscr{A}_r = (A_r,\delta_r,\tau_{a_0},\tau_r)$ na sledeći način: skup stanja $A_r$ je skup svih desnih jezika povezanih sa stanjima iz $\mathscr{A}$, i $\delta_r : A_r \times X \to A_r$ i $\tau_r : A_r \to L$ su dati sa:

$$\delta_r(\tau_a,x) = \tau_{\delta(a,x)}, \qquad \tau_r(\tau_a) = \tau_a(\varepsilon),$$

za sve $\tau_a \in A_r$. Imamo sledeće:

**Teorema 2.4.** *Neka je $\mathscr{A} = (A,\delta,a_0,\tau)$ dostižan deterministički fazi automat. Tada je $\mathscr{A}_r$ dostižan deterministički fazi automat izomorfan derivatnom automatu $\mathscr{A}_f$ fazi jezika $f = [\![\mathscr{A}]\!]$.*

Automat $\mathscr{A}_r$ se zove *automat desnog jezika* od $\mathscr{A}$.
Iz predhodne teoreme dobijamo sledeću posledicu.

**Lema 2.4.** *Neka je $\mathscr{A} = (A,\delta,a_0,\tau)$ dostižan deterministički fazi automat. Ako su svi desni fazi jezici povezani sa stanjima od $\mathscr{A}$ medjusobno različiti, tada je $\mathscr{A}$ minimalan.*

Primetimo da ako je $\mathscr{A} = (A,\delta,a_0,\tau)$ krisp deterministički fazi automat, tada je njegov reverzni Nerodov automat $\mathscr{A}_{\overline{N}} = (A_{\overline{N}},\delta_{\overline{N}},\tau_e,\tau_{\overline{N}})$, gde je $A_{\overline{N}}$ i $\delta_{\overline{N}} : A_{\overline{N}} \times X \to A_{\overline{N}}$ ima istu formu kao u opštem slučaju, dok je funkcija $\tau_{\overline{N}} : A_{\overline{N}} \to L$ data sa

$$\tau_{\overline{N}}(\tau_u) = \tau_u(a_0), \tag{B.26}$$

za sve $u \in X^*$.

**Teorema 2.5.** *Za svaki dostižan deterministički fazi automat $\mathscr{A} = (A, \delta, a_0, \tau)$, reverzni Nerodov automat $\mathscr{A}_{\overline{N}}$ je minimalan deterministički fazi automat ekvivalentan sa $\overline{\mathscr{A}}$.*

Neka je $\mathscr{A}$ fazi automat nad alfabetom $X$ i kompletnom reziduiranom mrežom $\mathscr{L}$. *Automat Brzozowskog* od $\mathscr{A}$, u oznaci $\mathscr{A}_B$, je deterministički fazi automat dobijen iz $\mathscr{A}$ kada se dva puta primeni konstrukcija reverznog Nerodovog automata, tj.,

$$\mathscr{A}_B = \left(\mathscr{A}_{\overline{N}}\right)_{\overline{N}} = \left(\overline{(\overline{\mathscr{A}})_N}\right)_N.$$

U nastavku predstavljamo glavni rezultat ovog odeljka.

**Teorema 2.6.** *Neka je $\mathscr{A}$ fazi automat nad alfabetom $X$ i kompletnom reziduiranom mrežom $\mathscr{L}$. Automat Brzozowskog $\mathscr{A}_B$ je minimalan deterministički fazi automat ekvivalentan sa $\mathscr{A}$.*

Kao što je rečeno, algoritam Brzozowski tipa za fazi automate je procedura koja konstruiše deterministički fazi automat $\mathscr{A}_B$ iz fazi automata $\mathscr{A}$ tako što se dva puta primeni konstrukcija reverznog Nerodovog automata. Zbog toga, umesto algoritma za konstrukciju automat Brzozowskog, mi dajemo algoritam za konstrukciju reverznog Nerodovog automata, i da bi konstruisali automat Brzozowskog fazi automata $\mathscr{A}$, potrebno je da dva puta primenimo ovaj algoritam.

**Algoritam 2.2. (*Konstrukcija reverznog Nerodovog automata $\mathscr{A}_{\overline{N}}$*)** Ulaz algoritma je konačan fazi automat $\mathscr{A} = (A, \sigma^A, \delta^A, \tau^A)$, i izlaz je krisp-determinističk fazi automat $\mathscr{A}_{\overline{N}} = (A_{\overline{N}}, \delta_{\overline{N}}, \tau_\varepsilon, \tau_{\overline{N}})$.

Graf prelaza od $\mathscr{A}_{\overline{N}}$ je konstruisan induktivno na sledeći način:

(A1) Koren stabla je $\tau_\varepsilon = \tau^A$, i stavljamo $T_0 = \{\tau_\varepsilon\}$ i $s(\tau_\varepsilon) = 1$, i računamo $\tau_{\overline{N}}(\tau_\varepsilon) = \tau_\varepsilon \circ \sigma^A$.

(A2) Nakon $i$tog koraka neka je stablo $T_i$ konstruisano, i čvorovi u $T_i$ su označeni sa 'zatvoren' ili 'ne-zatvoren'. Značenje ovih pojmova biće objašnjeno u nastavku.

(A3) U sledećem koraku konstruišemo stablo $T_{i+1}$ obogaćivanjem stabla $T_i$ na sledeći način: za svaki ne zatvoren list $\tau_u$ koji se javlja u $T_i$, gde je $u \in X^*$, i za svaki $x \in X$ dodajemo čvor $\tau_{xu} = \delta^A_x \circ \tau_u$ i granu iz $\tau_u$ u $\tau_{xu}$ označenu sa $x$. Simultano, proveravamo da li je $\tau_{xu}$ fazi skup koji je već konstruisan. Ako jeste, tj. $\tau_{xu}$ je jdenako nekom predhodno izračunatom $\tau_v$, označavamo $\tau_{xu}$ sa zatvoren i postavljamo $s(\tau_{xu}) = s(\tau_v)$. Inače, računamo vrednost $\tau_{\overline{N}}(\tau_{xu}) = \tau_{xu} \circ \sigma^A$ i postavljamo $s(\tau_{xu})$ za neoznačen ceo broj. Procedura se završava kada su svi listovi označeni sa zatvoren.

(A4) Kada je stablo prelaza $\mathscr{A}_{\overline{N}}$ konstruisano, brišemo brišemo sve znake zatvorenja i lepimo listove za unutrašnje čvorove sa istom oznakom. Dijagram koji se dobija je graf prelaza od $\mathscr{A}_{\overline{N}}$.

## 5. Simultana determinizacija i redukcija

Neka je $\mathscr{A} = (A, \sigma^A, \delta^A, \tau^A)$ fazi automat. Fazi relacija $\varphi$ na $A$ se naziva *desno invarijantna* ako

$$\varphi \circ \delta_x^A \leqslant \delta_x^A \circ \varphi, \qquad \text{za svako } x \in X, \tag{B.27}$$

$$\varphi \circ \tau^A \leqslant \tau^A, \tag{B.28}$$

i naziva se *slabo desno invarijantna* ako

$$\varphi \circ \tau_u^A \leqslant \tau_u^A, \qquad \text{za svako } u \in X^*. \tag{B.29}$$

Slično definišemo dualne koncepte. Fazi relacija $\varphi$ na $A$ je *levo invariantna* ako

$$\delta_x^A \circ \varphi \leqslant \varphi \circ \delta_x^A, \qquad \text{za sve } x \in X, \tag{B.30}$$

$$\sigma^A \circ \varphi \leqslant \sigma^A, \tag{B.31}$$

i $\varphi$ je *slabo levo invariantna* ako

$$\sigma_u^A \circ \varphi \leqslant \sigma_u^A, \qquad \text{za sve } u \in X^*. \tag{B.32}$$

Neka je $\mathscr{A} = (A, \sigma^A, \delta^A, \tau^A)$ fazi automat i $\varphi$ fazi relacija na $A$. Za svako $u \in X^*$ definišemo fazi skup $\varphi_u : A \to L$ induktivno, na sledeći način: za nepraznu reč $\varepsilon$ i sve $u \in X^*$ i $x \in X$ postavljamo

$$\varphi_\varepsilon = \sigma^A \circ \varphi, \qquad \varphi_{ux} = \varphi_u \circ \delta_x^A \circ \varphi \tag{B.33}$$

Jasno, ako je $u = x_1 \dots x_n$, gde je $x_1, \dots, x_n \in X$, tada

$$\varphi_u = \sigma^A \circ \varphi \circ \delta_{x_1}^A \circ \varphi \circ \dots \circ \delta_{x_n}^A \circ \varphi. \tag{B.34}$$

Sada postavljamo $A_\varphi = \{\varphi_u \mid u \in X^*\}$, i definišemo $\delta_\varphi : A_\varphi \times X \to A_\varphi$ i $\tau_\varphi : A_\varphi \to L$ kao što sledi:

$$\delta_\varphi(\varphi_u, x) = \varphi_{ux}, \qquad \tau_\varphi(\varphi_u) = \varphi_u \circ \tau^A, \tag{B.35}$$

za sve $u \in X^*$ i $x \in X$. Ako $\varphi_u = \varphi_v$, za neke $u, v \in X^*$, tada za sve $x \in X$ imamo da važi

$$\delta_\varphi(\varphi_u, x) = \varphi_{ux} = \varphi_u \circ \delta_x \circ \varphi = \varphi_v \circ \delta_x \circ \varphi = \varphi_{vx} = \delta_\varphi(\varphi_v, x),$$

i stoga je $\delta_\varphi$ dobro defininsana funkcija. Jasno je da je $\tau_\varphi$ takodje dobro definisana funkcija, i kao posledica toga, $\mathscr{A}_\varphi = (A_\varphi, \varphi_\varepsilon, \delta_\varphi, \tau_\varphi)$ dobro definisan deterministički fazi automat.

Glavni problem sa kojim se ovde srećemo jeste da se nadje fazi relacija $\varphi$ takva da je automat $\mathscr{A}_\varphi$ ekvivalentan sa originalnim automat $\mathscr{A}$.

**Teorema 2.7.** *Neka je $\mathscr{A} = (A, \sigma^A, \delta^A, \tau^A)$ fazi automat i $\varphi$ refleksivna slabo desno invariantna fazi relacija na $\mathscr{A}$. Tada je $\mathscr{A}_\varphi = (A_\varphi, \varphi_\varepsilon, \delta_\varphi, \tau_\varphi)$ dostižan deterministički fazi automat ekvivalentan sa $\mathscr{A}$.*

**Teorema 2.8.** *Neka je $\mathscr{A} = (A, \sigma^A, \delta^A, \tau^A)$ fazi automat i $\varphi$ slabo desno invariantno fazi kvazi-uredjenje na $\mathscr{A}$. Tada je automat $\mathscr{A}_\varphi$ izomorfan Nerodovom automatu afterset fazi automata $\mathscr{A}/\varphi$.*

U slučaju kada se radi sa desno invariantnim fazi kvazi uredjenjima, moguće je uporediti veličinu odgovarajućih automata.

**Teorema 2.9.** *Neka je $\mathscr{A} = (A, \sigma^A, \delta^A, \tau^A)$ fazi automat i $\varphi$ i $\phi$ desno invariantna fazi kvazi-uredjenja na $\mathscr{A}$ takva da $\varphi \leqslant \phi$. Tada je automat $\mathscr{A}_\phi$ homomorfna slika automata $\mathscr{A}_\varphi$ i $|\mathscr{A}_\phi| \leqslant |\mathscr{A}_\varphi|$.*

Primetimo da kad je $\varphi$ refleksivno slabo levo invariantno fazi kvazi uredjenje na fazi automotu $\mathscr{A}$, tada je $\mathscr{A}_\varphi$ Nerodov automat od $\mathscr{A}$, i ne dobijamo nikakvu novu konstrukciju. Osim toga, važi sledeće.

**Teorema 2.10.** *Neka je $\mathscr{A} = (A, \sigma^A, \delta^A, \tau^A)$ fazi automat i $\varphi$ slabo levo invariantno fazi kvazi uredjenje na $\mathscr{A}$. Tada je Nerodov automat afterset fazi automata $\mathscr{A}/\varphi$ je izomorfan Nerodovom automatu od $\mathscr{A}$.*

Pokazaćemo da slabo levo invariantna fazi relacija može biti veoma korisna u drugoj konstrukciji, i može da se koristi u determinizaciji reverznog fazi automata od $\mathscr{A}$.

Neka je $\mathscr{A} = (A, \sigma^A, \delta^A, \tau^A)$ fazi automat i $\psi$ fazi relacija na $A$. Za svako $u \in X^*$ definišemo fazi skup $\psi^u : A \to L$ induktivno, na sledeći način: za praznu reč $\varepsilon$ i sve $u \in X^*$ i $x \in X$ postavljamo

$$\psi^\varepsilon = \psi \circ \tau^A, \qquad \psi^{xu} = \psi \circ \delta_x^A \circ \psi^u \tag{B.36}$$

Jasno, ako je $u = x_1 \dots x_n$, gde je $x_1, \dots, x_n \in X$, tada

$$\psi^u = \psi \circ \delta_{x_1}^A \circ \psi \circ \dots \circ \delta_{x_n}^A \circ \psi \circ \tau^A. \tag{B.37}$$

Sada stavljamo $A^\psi = \{\psi_u \mid u \in X^*\}$, i definišemo $\delta^\psi : A^\psi \times X \to A^\psi$ i $\tau^\psi : A^\psi \to L$ na sledeći način:

$$\delta^\psi(\psi^u, x) = \psi^{xu}, \qquad \tau^\psi(\psi^u) = \sigma^A \circ \psi^u, \tag{B.38}$$

za sve $u \in X^*$ i $x \in X$. Ako je $\psi_u = \psi_v$, za neke $u, v \in X^*$, tada za sve $x \in X$ imamo da je

$$\delta^{\psi}(\psi^u, x) = \psi^{xu} = \psi \circ \delta_x^A \circ \psi^u = \psi \circ \delta_x^A \circ \psi^v = \psi^{xv} = \delta^{\psi}(\psi^v, x),$$

i stoga, $\delta^{\psi}$ je dobro definisana funkcija. Jasno, $\tau^{\psi}$ je dobro definisana funkcija, pa je $\mathscr{A}^{\psi} = (A^{\psi}, \psi^{\varepsilon}, \delta^{\psi}, \tau^{\psi})$ dobro definisan deterministički fazi automat.

**Teorema 2.11.** *Neka je $\mathscr{A} = (A, \sigma^A, \delta^A, \tau^A)$ fazi automat i $\psi$ slabo levo invariantno fazi relacija $\mathscr{A}$. Tada je $\mathscr{A}^{\psi} = (A^{\psi}, \psi^{\varepsilon}, \delta^{\psi}, \tau^{\psi})$ dostižan deterministički fazi automat ekvivalentan reverznom fazi automat od $\mathscr{A}$.*

**Teorema 2.12.** *Neka je $\mathscr{A} = (A, \sigma^A, \delta^A, \tau^A)$ fazi automat i $\psi$ slabo levo invariantno fazi kvazi uredjenje na $\mathscr{A}$. Tada je automat $\mathscr{A}^{\psi}$ izomorfan reverznom Nerodovom automatu od afterset fazi automata $\mathscr{A}/\psi$.*

**Teorema 2.13.** *Neka je $\mathscr{A} = (A, \sigma^A, \delta^A, \tau^A)$ fazi automat i $\psi$ i $\psi'$ levo invariantna fazi kvazi uredjenja na $\mathscr{A}$ takva da $\psi \leqslant \psi'$. Tada je automat $\mathscr{A}^{\psi'}$ homomorfna slika automata $\mathscr{A}^{\psi}$.*

Primetimo dae reverzni Nerodov automat igra važnu ulogu u determinizaciji Brzozowski tipa za fazi automate. Naime, pokazano je u [60] da kada se podje od fazi automata $\mathscr{A}$, i dva puta uzastopno primeni konstrukcija reverznog Nerodovog automata proizvodi minimalni deterministički fazi automat koji je ekvivalentan sa $\mathscr{A}$. Pokazaćemo da prva od ove dve konstrukcije može biti zamenjena konstrukcijom automata $\mathscr{A}^{\psi}$, za neku refleksivnu slabo levo invariantnu fazi relaciju $\psi$ na $\mathscr{A}$.

**Teorema 2.14.** *Neka je $\mathscr{A} = (A, \sigma^A, \delta^A, \tau^A)$ fazi automat i $\psi$ refleksivna slabo levo invariantna fazi relacija na $\mathscr{A}$. Tada je reverzni Nerodov automat od $\mathscr{A}^{\psi}$ minimal deterministički fazi automat ekvivalentan sa $\mathscr{A}$.*

Kako automat $\mathscr{A}^{\psi}$ može biti značajno manji od reverznog Nerodovog automata $\mathscr{A}_{\overline{N}}$ zamenom $\mathscr{A}_{\overline{N}}$ sa $\mathscr{A}^{\psi}$ u prvom koraku procedura tipa Brzozowskog možemo smanjiti porast broja stanja koji se može javiti u tom koraku. U drugom koraku, takav problem ne postoji jer konstrukcija datog minimalnog determinističkog fazi automata ekvivalentnog sa $\mathscr{A}$.

Neka je $\mathscr{A} = (A, \sigma^A, \delta^A, \tau^A)$ fazi automat nad alfabetom $X = \{x_1, \ldots, x_m\}$ i $\varphi$ fazi relacija na $A$. Za svako $u \in X^*$ definisemo $(m+1)$-torku $\varphi_u^c$ sa

$$\varphi_u^c = (\varphi_{ux_1}, \ldots, \varphi_{ux_m}, \varphi_u \circ \tau^A) = (\varphi_u \circ \delta_{x_1}^A, \ldots, \varphi_u \circ \delta_{x_m}^A, \varphi_u \circ \tau^A),$$

postavljamo $A_{\varphi}^c = \{\varphi_u^c \mid u \in X^*\}$, i definišemo $\delta_{\varphi}^c : A_{\varphi}^c \times X \to A_{\varphi}^c$ i $\tau_{\varphi}^c : A_{\varphi}^c \to L$ na sledeći način:

$$\delta_{\varphi}^c(\varphi_u^c, x) = \varphi_{ux}^c, \qquad \tau_{\varphi}^c(\varphi_u^c) = \varphi_u \circ \tau^A, \tag{B.39}$$

za sve $u \in X^*$ i $x \in X$.

**Teorema 2.15.** *Neka je* $\mathscr{A} = (A, \sigma^A, \delta^A, \tau^A)$ *fazi automat nad alfabetom* $X = \{x_1, \ldots, x_m\}$ *i* $\varphi$ *slabo desno invariantna fazi relacija na* $A$. *Tada je* $\mathscr{A}_\varphi^c = (A_\varphi^c, \varphi_\varepsilon^c, \delta_\varphi^c, \tau_\varphi^c)$ *dostižan deterministički fazi automat ekvivalentan sa* $\mathscr{A}$.

U slučaju kada je $\varphi$ obična jednakost na $A$, imamo $\varphi_u = \sigma_u^A$, za svako $u \in X^*$, i $\mathscr{A}_\varphi^c$ je automat konstruisan u [61], *redukovani Nerodov automaton* od $\mathscr{A}$. Ovde ćemo koristiti drugu terminiologiju. Naime, prvih $m$ elemenata u $(m+1)$-torci $\varphi_u^c$ su deca čvora $\varphi_u$ u grafu prelaza automata $\mathscr{A}_\varphi$, i $m+1$-i element od $\varphi_u^c$ odredjuje stepen od $\varphi_u$ u $\mathscr{A}_\varphi$. Zbog toga će automat $\mathscr{A}_\varphi^c$ biti nazvan *children automat* od $\mathscr{A}_\varphi$.

**Teorema 2.16.** *Neka je* $\mathscr{A} = (A, \sigma^A, \delta^A, \tau^A)$ *fazi automat nad alfabetom* $X = \{x_1, \ldots, x_m\}$ *i neka su* $\varphi$ *i* $\phi$ *desno invariantna fazi kvazi uredjenja na* $A$ *takva da* $\varphi \leqslant \phi$. *Tada je automat* $\mathscr{A}_\phi^c$ *homomorfna slika automata* $\mathscr{A}_\varphi^c$, *i kao posledica toga,* $|\mathscr{A}_\phi^c| \leqslant |\mathscr{A}_\varphi^c|$.

**Teorema 2.17.** *Neka je* $\mathscr{A} = (A, \sigma^A, \delta^A, \tau^A)$ *fazi automat nad* $X = \{x_1, \ldots, x_m\}$ *i neka je* $\varphi$ *slabo desno invariantno fazi kvazi uredjenje na* $\mathscr{A}$. *Tada je automat* $\mathscr{A}_\varphi^c$ *izomorfan automatu* $\mathscr{B}_N^c$, *gde je* $\mathscr{B} = \mathscr{A}/\varphi$ *afterset fazi automat od* $\mathscr{A}$ *u odnosu na* $\varphi$.

**Teorema 2.18.** *Neka je* $\mathscr{A} = (A, \sigma^A, \delta^A, \tau^A)$ *fazi automat nad* $X = \{x_1, \ldots, x_m\}$ *i neka je* $\varphi$ *slabo desno invariantno fazi kvazi uredjenje na* $A$. *Tada je automat* $\mathscr{A}_\varphi^c$ *homomorfna slika automata oba automata* $\mathscr{A}_\varphi$ *i* $\mathscr{A}_N^c$, *i stoga važi,* $|\mathscr{A}_\varphi^c| \leqslant |\mathscr{A}_\varphi|$ *i* $|\mathscr{A}_\varphi^c| \leqslant |\mathscr{A}_N^c|$.

**Algoritam 2.3.** ***(Konstrukcija automata*** $\mathscr{A}_\varphi$***)*** Ulaz algoritma je fazi konačan automat $\mathscr{A} = (A, \sigma^A, \delta^A, \tau^A)$ sa $n$ stanja, nad konačnim alfabetom $X$ sa $m$ slova, i fazi relacija $\varphi$ na $A$, i izlaz je deterministički fazi automat $\mathscr{A}_\varphi = (A_\varphi, \varphi_\varepsilon, \delta_\varphi, \tau_\varphi)$.

Graf prelaza od $\mathscr{A}_\varphi$ je konstruisan induktivno na sledeći način:

(A1) Koren stabla je $\varphi_\varepsilon = \sigma^A \circ \varphi$, i stavljamo $T_0 = \{\varphi_\varepsilon\}$ i $s(\varphi_\varepsilon) = 1$, računamo $\tau_\varphi(\varphi_\varepsilon) = \varphi_\varepsilon \circ \tau^A$.

(A2) Nakon $i$tog koraka neka je stablo $T_i$ konstruisano, i čvorovi u $T_i$ su označeni sa 'zatvoren' ili 'ne-zatvoren'. Značenje ovih pojmova biće objašnjeno u nastavku.

(A3) U sledećem koraku konstruišemo stablo $T_{i+1}$ obogaćivanjem stabla $T_i$ na sledeći način: za svaki ne zatvoren list $\varphi_u$ koji se javlja u $T_i$, gde je $u \in X^*$, i za svaki $x \in X$ dodajemo čvor $\varphi_{ux} = \varphi_u \circ \delta_x^A \circ \varphi$ i granu iz $\varphi_u$ u $\varphi_{ux}$ označenu sa $x$. Simultano, proveravamo da li je $\varphi_{ux}$ fazi skup koji je već konstruisan. Ako jeste, tj. ,ako je $\varphi_{ux}$ je jednako nekom predhodno izračunatom $\varphi_v$, označimo $\varphi_{ux}$ kao zatvoren i stavljamo $s(\varphi_{ux}) = s(\varphi_v)$. Inače, računamo vrednost $\tau_\varphi(\varphi_{ux}) = \varphi_{ux} \circ \tau^A$ i stavljamo $s(\varphi_{ux})$ za neoznačen ceo broj. Procedura se završava kada su svi listovi označeni kao zatvoreni.

(A4) Kada je stablo prelaza $\mathscr{A}_\varphi$ konstruisano, brišemo brišemo sve znake zatvorenja i lepimo listove za unutrašnje čvorove sa istom oznakom. Dijagram koji se dobija je graf prelaza od $\mathscr{A}_\varphi$.

**Algoritam 2.4.** ***(Konstrukcija automata $\mathscr{A}^\psi$)*** Ulaz algoritma je fazi konačan automat $\mathscr{A} = (A, \sigma^A, \delta^A, \tau^A)$ sa $n$ stanja, nad konačnim alfabetom $X$ sa $m$ slova, i fazi relacija $\psi$ na $A$, i izlaz je deterministički fazi automat $\mathscr{A}^\psi = (A^\psi, \psi^\varepsilon, \delta^\psi, \tau^\psi)$.

Graf prelaza od $\mathscr{A}^\psi$ je konstruisan induktivno na sledeći način:

(A1) Koren stabla je $\psi^\varepsilon = \psi \circ \tau^A$, i stavljamo $T_0 = \{\psi^\varepsilon\}$ i $s(\psi^\varepsilon) = 1$, i računamo vrednost $\tau^\psi(\psi^\varepsilon) = \sigma^A \circ \psi^\varepsilon$.

(A2) Nakon $i$tog koraka neka je stablo $T_i$ konstruisano, i čvororvi u $T_i$ su označeni sa 'zatvoren' ili 'ne-zatvoren'. Značenje ovih pojmova biće objašnjeno u nastavku.

(A3) U sledećem koraku konstruišemo stablo $T_{i+1}$ obogaćivanjem stabla $T_i$ na sledeći način: za svaki ne zatvoren list $\psi^u$ u $T_i$, gde je $u \in X^*$, i za svaki $x \in X$ dodajemo čvor $\psi^{xu} = \psi \circ \delta_x^A \circ \psi^u$ i granu iz $\psi^u$ u $\psi^{xu}$ označenu sa $x$. Istovremeno, proveravamo da li je $\psi^{xu}$ fazi skup koji je već konstruisan. Ako jeste, tj. ,ako je $\psi^{xu}$ je jednako nekom predhodno izračunatom $\psi^v$, označavamo $\psi^{xu}$ sa zatvoren i stavljamo $s(\psi^{xu}) = s(\psi^v)$. Inače, računamo vrednost $\tau^\psi(\psi^{xu}) = \sigma^A \circ \psi^{xu}$ i stavljamo $s(\psi^{xu})$ za neoznačen ceo broj. Procedura se završava kada su svi listovi označeni sa zatvoren.

(A4) Kada je stablo prelaza $\mathscr{A}^\psi$ konstruisano, brišemo brišemo sve znake zatvorenja i lepimo listove za unutrašnje čvorove sa istom oznakom. Dijagram koji se dobija je graf prelaza od $\mathscr{A}^\psi$.

U nastavku predstavljamo algoritme za računanje najvećeg desno i slabo desno invariantnog kvazi uredjenja.

**Algoritam 2.5.** ***(Izračunavanje najvećeg desno invarijantnog fazi kvazi-uredjenja)*** Ulaz algoritma je konačan fazi automat $\mathscr{A} = (A, \sigma^A, \delta^A, \tau^A)$ sa $n$ stanja, nad alfabetom $X$ sa $m$ slova. Algoritam računa najvećeg desno invariantno kvazi uredjenje $\varphi^{\mathrm{ri}}$ na $\mathscr{A}$.

Procedura predstavlja računanje sekvence $\{\varphi_k\}_{k\in\mathbb{N}}$, na sledeći način:

(A1) U prvom koraku stavljamo $\varphi_1 = \tau^A / \tau^A$.

(A2) Nakon $k$tog koraka fazi kvazi uredjenje $\varphi_k$ je konstruisano.

(A3) U narednom koraku konstruišemo $\varphi_{k+1}$ pomoću formule

$$\varphi_{k+1} = \varphi_k \wedge \Big[ \bigwedge_{x\in X} (\delta_x^A \circ \varphi_k) / (\delta_x^A \circ \varphi_k) \Big]. \tag{B.40}$$

(A4) Simultano, proveravamo da li je $\varphi_{k+1} = \varphi_k$.

(A5) Kada nadjemo najmanji $s$ takav da je $\varphi_{s+1} = \varphi_s$, procedura az konstrukciju sekvence $\{\varphi_k\}_{k\in\mathbb{N}}$ se završava i $\varphi^{\mathrm{ri}} = \varphi_s$.

Ovaj algoritam se izvršava u polinomijalnom vremenu.

**Algoritam 2.6.** ***(Izračunavanje najvećeg slabo desno invariantnog fazi kvazi-uredjenja)*** Ulaz algoritma je konačan fazi automat $\mathscr{A} = (A, \sigma^A, \delta^A, \tau^A)$ sa $n$ stanja, nad konačnim alfabetom $X$ sa $m$ slova. Algoritam računa najveće slabo desno invariantno kvazi uredjenje $\varphi^{\mathrm{wri}}$ na $\mathscr{A}$.

Procedura je sačinjena od sledećih koraka:

(A1) Prvo računamo $\{\tau_u^A \mid u \in X^*\}$, pomoću Algoritma 2.4..
(A2) Zatim računamo $\varphi^{\mathrm{wri}}$ pomoću formule

$$\varphi^{\mathrm{wri}} = \bigwedge_{u \in X^*} \tau_u^A / \tau_u^A.$$

**Algoritam 2.7.** ***(Konstrukcija children automata $\mathscr{A}_\varphi^c$)*** Ulaz algoritma je konačan fazi automat $\mathscr{A} = (A, \sigma^A, \delta^A, \tau^A)$ sa $n$ stanja, nad konačnim alfabetom $X = \{x_1, \ldots, x_m\}$ sa $m$ slova, i izlaz je children automat $\mathscr{A}_\varphi^c = (A_\varphi^c, \varphi_\varepsilon^c, \delta_\varphi^c, \tau_\varphi^c)$, gde je $\varphi$ najveće slabo desno invariantno fazi kvazi-uredjenje ili najveće desno invariantno fazi kvazi-uredjenje $\mathscr{A}$.

Graf prelaza $\mathscr{A}_\varphi$ i graf prelaza $\mathscr{A}_\varphi^c$ su konstruisani na sledeći način:

(A1) Računamo $\varphi$ pomoću Algoritma 2.6. i 2.5., i konstruišemo $T$ od $\mathscr{A}_\varphi$ pomoću Algoritma 2.3..
(A2) Za svaki nezatvoren čvor $\varphi_u$ stabla $T$ označavamo čvor $\varphi_u^c$ grafa $G$ na sledeći način: Kada su sva deca $\varphi_{ux_1}, \ldots, \varphi_{ux_m}$ od $\varphi_u$ u stablu $T$ formirana, formiramo čvor $\varphi_u^c = (\varphi_{ux_1}, \ldots, \varphi_{ux_m}, \varphi_u \circ \tau^A)$ u grafu $G$. Simultano, proveravamo da li je $\varphi_u^c$ $(m+1)$-torka koja je već konstruisana. Ako jeste tj. ako je $\varphi_u^c$ jednak nekom predhodno izračunatom $\varphi_v^c$, označavamo $\varphi_u^c$ sa zatvoren i stavljamo $t(\varphi_u^c) = t(\varphi_v^c)$. Inače, stavljamo $\tau_\varphi^c(\varphi_u^c) = \varphi_u \circ \tau^A$ i $t(\varphi_u^c)$ postavljamo da bude sledeći ceo broj koji nije korišćen kao vrednost za $t(\cdot)$.
(A3) Za svaki nezatvoren čvor $\varphi_u^c$ grafa $G$ i svaki $x \in X$, ako je $\varphi_v$ nezatvoren čvor u $T$ takav da je $s(\varphi_{ux}) = s(\varphi_v)$, u graf $G$ dodajemo granu iz $\varphi_u^c$ u $\varphi_v^c$ označenu sa $x$.
(A4) Kada je graf $G$ konstruisan, konstruisan, brišemo brišemo sve znake zatvorenja i lepimo listove za unutrašnje čvorove sa istom oznakom. Dijagram koji se dobija je graf prelaza od $\mathscr{A}_\varphi^c$.

## 6. Determinizacija pomoću stepena jezičke inkluzije

Neka je $\mathscr{A} = (A, \sigma, X, \delta, \tau)$ fazi automat nad kompletnom reziduiranom mrežom $\mathscr{L}$. *Jezik stanja* $a \in A$ fazi automata $\mathscr{A}$ je fazi jezik $\tau_a \in \mathscr{F}(X^*)$ definisan sa:

$$\tau_a(u) = \bigvee_{b \in A} \delta^*(a, u, b) \otimes \tau(b),$$

za sve $u \in X^*$, ili ekvivalentno fazi jezik raspoznat fazi automatom $\mathscr{A}' = (A, a, X, \delta, \tau)$ dobijenim iz $\mathscr{A}$ kada se postavi da $a$ bude krisp inicijalno stanje. Analogno, definišemo *jezik fazi skupa* $\mu \in \mathscr{F}(A)$ od $\mathscr{A}$ kao fazi jezik $\tau_\mu \in \mathscr{F}(X^*)$ dat sa

$$\tau_\mu(u) = \bigvee_{a,b\in A} \mu(a)\otimes\delta^*(a,u,b)\otimes\tau(b),$$

za sve $u \in X^*$, tj. fazi jezik raspoznat fazi automatom $\mathscr{A}_\mu = (A,\mu,X,\delta,\tau)$ dobijen iz $\mathscr{A}$ kada se $\mu$ postavi za fazi skup inicijanih stanja. Jasno, $[\![\mathscr{A}]\!] = \tau_\sigma$, tj. jezik fazi skupa$\sigma$ od $\mathscr{A}$ je ekvivalentan jeziku raspoznatom sa $\mathscr{A}$.

Stepen inkluzije dva fazi skupa $\mu,\nu \in \mathscr{F}(A)$ je uveden ne sledeći način

$$I(\mu,\nu) = \bigwedge_{a\in A} \mu(a) \to \nu(a).$$

Slično, za dat fazi automat $\mathscr{A}$, stanja $a,b \in A$ i fazi skupove $\mu,\nu \in \mathscr{F}(A)$, definišemo *stepen jezičke inkluzije* od $a$ i $b$ u $\mathscr{A}$ sa:

$$L(a,b) = \bigwedge_{u\in X^*} \tau_a(u) \to \tau_b(u),$$

*stepen jezičke inkluzije* od $a$ i $\mu$ u $\mathscr{A}$ sa

$$L(a,\mu) = \bigwedge_{u\in X^*} \tau_a(u) \to \tau_\mu(u),$$

i *stepen jezičke inkluzije* od $\mu$ i $\nu$ u $\mathscr{A}$ sa

$$L(\mu,\nu) = \bigwedge_{u\in X^*} \tau_\mu(u) \to \tau_\nu(u).$$

Neka je $\mathscr{A} = (A,\sigma,X,\delta,\tau)$ fazi automat nad kompletnom reziduiranom mrežom $\mathscr{L}$. Definišemo fazi skupove $d_u \in \mathscr{F}(A)$, $u \in X^*$ na sledeći način: Neka je $e \in X^*$ prazna reč, $u \in X^*$ i $x \in X$, tada imamo:

$$d_\varepsilon(a) = L(a,\sigma) = \bigwedge_{v\in X^*} \tau_v(a) \to \sigma\circ\tau_v, \tag{B.41}$$

i

$$d_{ux}(a) = L(a, d_u\circ\delta_x) = \bigwedge_{v\in X^*} \tau_v(a) \to d_u\delta_x\sigma\circ\tau_v, \tag{B.42}$$

za sve $a \in A$. Označimo $A_d = \{d_u \mid u \in X^*\}$.

**Teorema 2.19.** *Neka je $\mathscr{A} = (A,\sigma,X,\delta,\tau)$ fazi automat nad kompletnom reziduiranom mrežom $\mathscr{L}$. Tada*

$$d_u\circ\tau_w = \sigma_u\circ\tau_w, \tag{B.43}$$

*za sve $u,w \in X^*$.*

Prema predhodnoj teoremi imamo

$$d_e(a) = \bigwedge_{v \in X^*} \tau_v(a) \to \sigma \circ \tau_v$$

i

$$d_{ux}(a) = \bigwedge_{v \in X^*} \tau_v(a) \to d_u \circ \delta_x \circ \tau_v = \bigwedge_{v \in X^*} \tau_v(a) \to \sigma_{ux} \circ \tau_v$$

za sve $u \in X^*$ i $x \in X$, i odatle

$$d_u(a) = \bigwedge_{v \in X^*} \tau_v(a) \to \sigma_u \circ \tau_v \tag{B.44}$$

važi za sve $u \in X^*$. Sada definišemo $\delta_d : A_d \times X \to A_d$ i $\tau_d : A_d \to L$ sa

$$\delta_d(d_u, x) = d_{ux}, \qquad \tau_d(d_u) = d_u \circ \tau,$$

za sve $d_u \in A_d$ i $x \in X$.

**Teorema 2.20.** *Neka je $\mathscr{A} = (A, \sigma, X, \delta, \tau)$ fazi automat nad $\mathscr{L}$. Tada je automat $\mathscr{A}_d = (A_d, \delta_d, d_\varepsilon, \tau_d)$ deterministički fazi automat,i ekvivalentan je sa $\mathscr{A}$, tj. $[\![\mathscr{A}_d]\!] = [\![\mathscr{A}]\!]$.*

**Teorema 2.21.** *Neka je $\mathscr{A} = (A, \sigma, X, \delta, \tau)$ fazi automat nad $\mathscr{L}$. Tada je $\mathscr{A}_d = (A_d, \delta_d, d_\varepsilon, \tau_d)$ minimalan deterministički fazi automat ekvivalentan sa $\mathscr{A}$.*

U nastavku dajemo algoritam za konstrukciju $\mathscr{A}_d$ datog fazi automata $\mathscr{A} = (A, \delta, \sigma, \tau)$.

**Algoritam 2.8. (*Konstrukcija automata $\mathscr{A}_d$*)** Ulaz algoritma je fazi automat $\mathscr{A} = (A, \delta, \sigma, \tau)$ nad $X$ i $L$, a izlaz kdka $\mathscr{A}_d = (A_d, d_\varepsilon, \delta_d, \tau_d)$ koji je minimalan dkfa ekvivalentan sa $\mathscr{A}$.

Graf prelaza od $\mathscr{A}_d$ se konstruiše induktivno na sledeći način:

(A1) Računamo sve $\tau_u$, $u \in X^*$ pomoću Algoritma 2.6..

(A2) Koren stabla je $d_\varepsilon$, izračunat pomocu formule (B.41), i stavljamo $S_0 = \{d_\varepsilon\}$.

(A3) Nakon $i$tog koraka neka je stablo $S_i$ konstruisano, i čvorovi u $T_i$ su označeni sa 'zatvoren' ili 'ne-zatvoren'. Značenje ovih pojmova biće objašnjeno u nastavku.

(A4) U sledećem koraku konstruišemo stablo $S_{i+1}$ obogaćivanjem stabla $S_i$ na sledeći način: za svaki ne zatvoren list $d_u$ u $S_i$, , gde je $u \in X^*$, i za svaki $x \in X$ dodajemo čvor $d_{ux}$, izračunat formulom (B.42). Nakon tog stavljamo granu iz $d_u$ u $d_{ux}$ i označavamo je sa $x$. U slučaju da je $d_{ux}$ stanje koje je već bilo konstruisano označavamo $d_{ux}$ sa *zatvoren*. Procedura se završava kada su svi listovi označeni sa zatvoren.

(A5) Simultano, za svaki ne zatvoren list $d_u$ u $S_i$, gde je $u \in X^*$, računamo vrednost $\tau_d(d_u) = d_u \circ \tau$,

(A6) Kada je stablo prelaza $\mathscr{A}_d$ konstruisano, brišemo sve znake zatvorenja i lepimo listove za unutrašnje čvorove sa istom oznakom. Dijagram koji se dobija je graf prelaza od $\mathscr{A}_d$.

# Appendix C
# Biography

Zorana Jančić was born on July 13, 1984. in Niš, Serbia. In 1999., she finished elementary school ”Branko Miljković” and gained ”Vuk Karadžić diploma” for excellent success. She finished high school ”Bora Stanković” in Niš with excellent marks. In the year 2003 she started her studies on the Faculty of Science and Mathematics, department of mathematics and informatics, University of Niš. She graduated in the year 2008., with the average grade 9,82.

She received ”city of Niš” scholarship for talented students in the year 2006. and 2008. and ”Eurobank EFG scholarship” for 100 top students in the year 2007. In July 2007., she got a scholarship, ”Travelling to Europe” for the best final-year students of the Republic of Serbia. In October 2008., she started PhD studies at the department of Computer Science, Faculty of Science and Mathematics, University of Niš.

In February 2009. she started working at Faculty of Science and Mathematics in University of Niš, as a junior researcher on project ”Algebraic structures and methods for information processing” (144011). Since 2011., she is working on the project ”Development of methods for computation and information processing: theory and application” (174013).

She is currently working as teaching assistant at Faculty of Science and Mathematics. Her research interests include automata theory (in particular, determinization of weighted and fuzzy automata).

## LIST OF SCIENTIFIC PAPERS

### Papers published in international journals on the SCI list

1. Z. Jančić, J.Ignjatović, M. Ćirić, An improved algorithm for determinization of weighted and fuzzy automata, Information Sciences 181 (2011) 13581368.
2. Z. Jančić, M. Ćirić, Brzozowski type determinization for fuzzy automata, Fuzzy Sets and Systems (2014) http://dx.doi.org/10.1016/j.fss.2014.02.021

### Papers submitted for publication in international journals on the SCI list

3. Z. Jančić, I. Micić, J.Ignjatović, M. Ćirić, Two-in-one: determinization and reduction of fuzzy automata, submitted to Information Sciences.
4. Z. Jančić, I. Micić, J.Ignjatović, M. Ćirić, Determinization of fuzzy automata by means of the degree of language inclusion, submitted to IEEE Transactions on Fuzzy Systems.
5. J. Ignjatović, M. Ćirić, Z. Jančić, Weighted automata with output, submitted to Soft Computing.

### Participation in international conferences with published list of abstracts

6. Z. Jančić, I. Jančić, J. Ignjatović, M. Ćirić, Fuzzy and weighted automata: Canonization methods, International Workshop on Weighted Automata: Theory and Applications, WATA 2012, Dresden, Germany, May 29-June 2, 2012.
7. Z. Jančić, J. Ignjatović, M. Ćirić, Fuzzy and weighted automata: determinization methods, International Workshop on Weighted Automata: Theory and Applications, WATA 2010, Leipzig, Germany, May 3-7, 2010.
8. Z. Jančić, J. Ignjatović, M. Ćirić, Fuzzy and weighted automata: determinization methods, The 3rd Novi Sad Algebraic Conference, Novi Sad, Serbia, August 17-21, 2009.

# References


1. L. Aceto, A. Ingolfsdottir, K. G. Larsen, J. Srba, Reactive Systems: Modelling, Specification and Verification, Cambridge University Press, Cambridge, 2007.
2. N. C. Basak, A. Gupta, On quotient machines of a fuzzy automaton and the minimal machine, Fuzzy Sets and Systems 125 (2002), 223–229.
3. A. L. Buchsbaum, R. Giancarlo, J.R. Westbrook, On the determinization of weighted finite automata, SIAM Journal on Computing 30 (5) (2000) 1502–1531.
4. M. P. Béal, S. Lombardy, J. Sakarovitch, On the equivalence of $\mathbb{Z}$-automata, In: L. Caires et al. (eds.), ICALP 2005, Springer, Heidelberg, Lecture Notes in Computer Science 3580 (2005) 397–409.
5. M. P. Béal, S. Lombardy, J. Sakarovitch, Conjugacy and equivalence of weighted automata and functional transducers. In: D. Grigoriev, J. Harrison, and E. A. Hirsch (eds.), CSR 2006, Springer, Heidelberg, Lecture Notes in Computer Science 3967 (2006) 58–69.
6. M. P. Béal, D. Perrin, On the generating sequences of regular languages on $k$ symbols, Journal of the ACM 50 (2003) 955–980.
7. J. A. Brzozowski, Canonical regular expression and minimal state graphs for definite events, Mathematical Theory of automata, MRI Symposia Series, Polytechnic Press, Polytechnic Institute of Brooklyn, NY, 12 (1962) 529-561.
8. S. Bozapalidis, O. Louscou-Bozapalidou, On the recognizability of fuzzy languages I, Fuzzy Sets and Systems 157 (2006) 2394–2402.
9. S. Bozapalidis, O. Louscou-Bozapalidou, On the recognizability of fuzzy languages II, Fuzzy Sets and Systems 159 (2008) 107–113.
10. S. Bozapalidis, O. Louscou-Bozapalidou, Fuzzy tree language recognizability, Fuzzy Sets and Systems 161 (2010) 716–734.
11. R. Bělohlávek, Fuzzy Relational Systems: Foundations and Principles, Kluwer, New York, 2002.
12. R. Bělohlávek, Determinism and fuzzy automata, Information Sciences 143 (2002) 205–209.
13. R. Bělohlávek, V. Vychodil, Fuzzy Equational Logic, Springer, Berlin/Heidelberg, 2005.
14. S. L. Bloom, Z. Ésik, Iteration Theories: The Equational Logic of Iterative Processes, EATCS Monographs on Theoretical Computer Science, Springer, Berlin-Heilderberg, 1993.
15. J. Berstel, Ch. Reutenauer, Rational Series and Their Languages, EATCS Monograph on Theoretical Computer Science, vol. 12, Springer-Verlag, 1988.

16. T. Brihaye, Words and bisimulations of dynamical systems, Discrete Mathematics and Theoretical Computer Science 9 (2) (2007) 11–32.
17. P. Buchholz, Bisimulation relations for weighted automata, Theoretical Computer Science 393 (2008) 109–123.
18. Y. Cao, G. Chen, E. Kerre, Bisimulations for fuzzy transition systems, IEEE Transactions on Fuzzy Systems 19 (2011) 540-552.
19. Y. Z. Cao, M. S. Ying, Supervisory control of fuzzy discrete event systems, IEEE Transactions on Systems, Man, and Cybernetics – Part B 35 (2005) 366–371.
20. Y. Z. Cao, M. S. Ying, Observability and decentralized control of fuzzy discrete-event systems, IEEE Transactions Fuzzy Systems 14 (2006) 202–216.
21. Y. Z. Cao, M. S. Ying, G. Q. Chen, State-based control of fuzzy discrete-event systems, IEEE Transactions on Systems, Man, and Cybernetics – Part B 37(2007) 410–424.
22. C. G. Cassandras, S. Lafortune, Introduction to Discrete Event Systems, Springer, 2008.
23. K. Chatterjee, L. Doyen, T. Henzinger, Quantitative languages, in: Proceedings of the 17th International Conference on Computer Science Logic (CSL), Lecture Notes in Computer Science 5213 (2008) 385–400.
24. K. Chatterjee, L. Doyen, T. Henzinger, Alternating weighted automata, in: Proceedings of the 17th International Symposium on Fundamentals of Computation Theory (FCT), Lecture Notes in Computer Science 5699 (2009) 3–13.
25. K. Chatterjee, L. Doyen, T. Henzinger, Expressiveness and closure properties for quantitative languages, in: Proceedings of LICS 2009: Logic in Computer Science, IEEE Computer Society Press, 2009, pp. 199–208.
26. W. Cheng, Z. Mo, Minimization algorithm of fuzzy finite automata, Fuzzy Sets and Systems 141 (2004), 439–448.
27. M. Ćirić, M. Droste, J. Ignjatović, H. Vogler, Determinization of weighted finite automata over strong bimonoids, Information Sciences 180 (2010) 3497–3520.
28. M. Ćirić, J. Ignjatović, S. Bogdanović, Uniform fuzzy relations and fuzzy functions, Fuzzy Sets and Systems 160 (2009) 1054–1081.
29. M. Ćirić, J. Ignjatovic, M. Bašić, I. Jančić, Nondeterministic automata: Simulation, bisimulation and structural equivalence, Information Sciences 261 (2014), 185–218.
30. M. Ćirić, J. Ignjatović, N. Damljanović, M. Bašić, Bisimulations for fuzzy automata, Fuzzy Sets and Systems 186 (2012) 100–139.
31. M. Ćirić, J. Ignjatović, I.Jančić, N. Damljanović, Computation of the greatest simulations and bisimulations between fuzzy automata, Fuzzy Sets and Systems 208 (2012) 22–42.
32. M. Ćirić, A. Stamenković, J. Ignjatović, T. Petković, Factorization of fuzzy automata, In: Csuhaj-Varju, E., Ésik, Z. (eds.), FCT 2007, Springer, Heidelberg, Lecture Notes in Computer Science 4639 (2007) 213–225.
33. M. Ćirić, A. Stamenković, J. Ignjatović, T. Petković, Fuzzy relation equations and reduction of fuzzy automata, Journal of Computer and System Sciences 76 (2010) 609–633.
34. M. Demirci, Fuzzy functions and their applications, Journal of Mathematical Analysis and Applications 252 (2000) 495–517.
35. M. Demirci, Foundations of fuzzy functions and vague algebra based on many-valued equivalence relations, Part I: Fuzzy functions and their applications, International Journal of general Systems 32 (2) (2003) 123–155.
36. M. Demirci, A theory of vague lattices based on many-valued equivalence relations – I: general representation results, Fuzzy Sets and Systems 151 (2005) 437–472.
37. M. Droste, T. Stüber, H. Vogler, Weighted finite automata over strong bimonoids, Information Sciences 180 (2010) 156–166.
38. M. Droste, H. Vogler, Kleene and Büchi results for weighted automata and multi-valued logics over arbitrary bounded lattices, in: Y. Gao et al. (Eds.), DLT 2010, Lecture Notes in Computer Science, 6224, 2010, pp. 160172.

39. D. Dubois, H. Prade, Fuzzy Sets and Systems: Theory and Applications, Academic Press, New York, 1980.
40. D. Dubois, H. Prade (eds.), Fundamentals of Fuzzy Sets, The Handbooks of Fuzzy Sets Series, Vol. 1, Kluwer Academic Publishers, 2000.
41. M. M. Gupta, G. N. Saridis, B. R. Gaines, Fuzzy Automata and Decision Processes, North-Holland, New York, 1977.
42. Z. Ésik, W. Kuich, A generalization of Kozen's axiomatization of the equational theory of the regular sets, in: Words, semigroups, and transductions, World Scientific, River Edge, NJ, 2001, pp. 99–114.
43. Z. Ésik, A. Maletti, Simulation vs. Equivalence, CoRR abs/1004.2426 (2010).
44. A. Dovier, C. Piazza, A. Policriti, An efficient algorithm for computing bisimulation equivalence, Theoretical Computer Science 311 (2004) 221–256.
45. M. R. Garey, D. S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, Freeman, San Francisco,1979.
46. R. Gentilini, C. Piazza, A. Policriti, From bisimulation to simulation: coarsest partition problems, Journal of Automated Reasoning 31 (2003) 73–103.
47. P. Hájek, Mathematics of fuzzy logic, Kluwer, Dordrecht, 1998.
48. T. A. Henzinger, P. W. Kopke, A. Puri, P. Varaiya, What's decidable about hybrid automata? Journal of Computer and System Sciences 57 (1998) 94–124.
49. J. Högberg, A. Maletti, J. May, Backward and forward bisimulation minimisation of tree automata, in: J. Holub, J. Žďárek (eds.), IAA07, Springer, Heidelberg, Lecture Notes in Computer Science 4783 (2007) 109–121.
50. J. Högberg, A. Maletti, J. May, Backward and forward bisimulation minimisation of tree automata, Theoretical Computer Science 410 (2009) 3539–3552.
51. U. Höhle, Commutative, residuated $\ell$-monoids, in: U. Höhle and E. P. Klement (Eds.), Non-Classical Logics and Their Applications to Fuzzy Subsets, Kluwer Academic Publishers, Boston, Dordrecht, 1995, pp. 53–106.
52. B. Hrúz, M. C. Zhou, Modeling and control of discrete-event dynamical systems: with Petri nets and other tools, Springer, 2007.
53. J. Ignjatović, M. Ćirić, Formal power series and regular operations on fuzzy languages, Information Sciences 180 (2010) 1104–1120.
54. J. Ignjatović, M. Ćirić, S. Bogdanović, Determinization of fuzzy automata with membership values in complete residuated lattices, Information Sciences 178 (2008) 164–180.
55. J. Ignjatović, M. Ćirić, S. Bogdanović, Fuzzy homomorphisms of algebras, Fuzzy Sets and Systems 160 (2009), 2345–2365.
56. J. Ignjatović, M. Ćirić, S. Bogdanović, On the greatest solutions to weakly linear systems of fuzzy relation inequalities and equations, Fuzzy Sets and Systems 161 (2010) 3081–3113.
57. J. Ignjatović, M. Ćirić, S. Bogdanović, T. Petković, Myhill-Nerode type theory for fuzzy languages and automata, Fuzzy Sets and Systems 161 (2010) 1288-1324.
58. J. Ignjatović, M. Ćirić, Z. Jančić, Weighted automata with output, Soft Computing, submitted for publication.
59. I. Jančić, Weak bisimulations for fuzzy automata, Fuzzy Sets and Systems (2013), http://dx.doi.org/10.1016/j.fss.2013.10.006.
60. Z. Jančić, M. Ćirić, Brzozowski type determinization for fuzzy automata, Fuzzy Sets and Systems (2014), http://dx.doi.org/10.1016/j.fss.2014.02.021.
61. Z. Jančić, J. Ignjatović, M. Ćirić, An improved algorithm for determinization of weighted and fuzzy automata, Information Sciences 181 (2011) 1358–1368.
62. Z. Jančić, I. Micić, J. Ignjatović, M. Ćirić, Two-in-one: determinization and state reduction of fuzzy automata, Information Sciences, submitted for publication.
63. Z. Jančić, I. Micić, J. Ignjatović, M. Ćirić, Determinization of fuzzy automata by means of language inclusion degrees, IEEE Transactions on Fuzzy Systems, submitted for publication.

64. T. Jiang, B. Ravikumar, Minimal NFA problems are hard, SIAM J. Comput. 22 (6) (1993) 1117–1141.
65. D. Kirsten, I. Mäurer, On the determinization of weighted automata, Journal of Automata, Languages and Combinatorics 10 (2005) 287–312.
66. P. C. Kannellakis, S. A. Smolka, CCS expressions, finite state processes, and three problems of equivalence, Information and Computation 86 (1990) 43–68.
67. E. Kilic, Diagnosability of fuzzy discrete event systems, Information Sciences 178 (2008) 858–870.
68. G. J. Klir, B. Yuan, Fuzzy Sets and Fuzzy Logic, Theory and Application, Prentice-Hall, Englevood Cliffs, NJ, 1995.
69. D. C. Kozen, Automata and Computability, Springer, 1997.
70. Y. M. Li, Finite automata based on quantum logic and their determinization, CoRR abs/0712.4341 (2007).
71. E. T. Lee, L. A. Zadeh, Note on fuzzy languages, Information Sciences 1 (1969) 421–434.
72. Y. M. Li, W. Pedrycz, Fuzzy finite automata and fuzzy regular expressions with membership values in lattice ordered monoids, Fuzzy Sets and Systems 156 (2005) 68–92.
73. T. Lengauer, D. Theune, Unstructured path problems and the making of semirings, in: Algorithms and Data Structures, Proceedings WADS91, Ottawa, Canada, Lecture Notes in Computer Science 519 (1991) 189–200.
74. F. Lin, H. Ying, Fuzzy discrete event systems and their observability, in: Proceedings of the 2001 IFSA/NAFIP Conference, 2001, pp. 1271–1276.
75. F. Lin, H. Ying, Modeling and control of fuzzy discrete event systems, IEEE Transactions on Man, Systems and Cybernetics – Part B 32 (2002) 408–415.
76. F. Lin, H. Ying, R. D. MacArthur, J. A. Cohn, D. Barth-Jones, L. R. Crane, Decision making in fuzzy discrete event systems, Information Sciences 177 (2007) 3749–3763.
77. H. Lei, Y. M. Li, Minimization of states in automata theory based on finite lattice-ordered monoids, Information Sciences 177 (2007), 1413–1421.
78. J. P. Liu, Y. M. Li, The relationship of controllability between classical and fuzzy discrete-event systems, Information Sciences 178 (2008) 4142–4151.
79. L. Li, D. W. Qiu, Decidability of minimization of fuzzy automata, arXiv:1305.4255.
80. N. Lynch, F. Vaandrager, Forward and backward simulations: Part I. Untimed systems, Information and Computation 121 (1995), 214–233.
81. M. Mohri, Finite-state transducers in language and speech processing, Computational Linguistics 23 (2) (1997) 269–311.
82. M. Mohri, Weighted automata algorithms, in: M. Droste, W. Kuich, H. Vogler (Eds.), Handbook of Weighted Automata, Springer-Verlag, 2009, pp. 213–254.
83. R. Milner, A calculus of communicating systems, in G. Goos and J. Hartmanis (eds.), Lecture Notes in Computer Science, vol. 92, Springer, 1980.
84. R. Milner, Communication and Concurrency, Prentice-Hall International, 1989.
85. R. Milner, Communicating and Mobile Systems: the $\pi$-Calculus, Cambridge University Press, Cambridge, 1999.
86. J. N. Mordeson, D. S. Malik, Fuzzy Automata and Languages: Theory and Applications, Chapman & Hall/CRC, Boca Raton, London, 2002.
87. D. S. Malik, J. N. Mordeson, M. K. Sen, Minimization of fuzzy finite automata, Information Sciences 113 (1999) 323–330.
88. R. Paige, R. E. Tarjan, Three partition refinement algorithms, SIAM Journal of Computing 16 (1987) 973–989.
89. D. Park, Concurrency and automata on infinite sequences, in: P. Deussen (ed.), Proc. 5th GI Conf., Karlsruhe, Germany, Lecture Notes in Computer Science 104 (1981), Springer-Verlag, pp. 167–183.
90. K. Peeva, Finite L-fuzzy acceptors, regular L-fuzzy grammars and syntactic pattern recognition, International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems 12 (2004) 89–104.

91. K. Peeva, Finite L-fuzzy machines, Fuzzy Sets and Systems 141 (2004) 415–437.
92. K. Peeva, Y. Kyosev, Fuzzy Relational Calculus: Theory, Applications, and Software (with CD-ROM), in Series "Advances in Fuzzy Systems – Applications and Theory", Vol 22, World Scientific, 2004.
93. K. Peeva, Z. Zahariev, Computing behavior of finite fuzzy machines – Algorithm and its application to reduction and minimization, Information Sciences 178 (2008) 4152–4165.
94. T. Petkovic, Congruences and homomorphisms of fuzzy automata, Fuzzy Sets and Systems 157 (2006) 444–458.
95. A. Stamenković, M. Ćirić, J. Ignjatović, Reduction of fuzzy automata by means of fuzzy quasi-orders, Information Sciences (2014), doi: http://dx.doi.org/10.1016/j.ins.2014.02.028.
96. D. W. Qiu, Automata theory based on completed residuated lattice-valued logic (I), Science in China, Ser. F, 44 (6) (2001) 419–429.
97. D. W. Qiu, Automata theory based on completed residuated lattice-valued logic (II), Science in China, Ser. F, 45 (6) (2002) 442–452.
98. D. W. Qiu, Characterizations of fuzzy finite automata, Fuzzy Sets and Systems 141 (2004) 391–414.
99. D. Qiu, Supervisory control of fuzzy discrete event systems: a formal approach, IEEE Transactions on Systems, Man and Cybernetics – Part B 35 (2005) 72–88.
100. D. W. Qiu, Pumping lemma in automata theory based on complete residuated lattice-valued logic: A note, Fuzzy Sets and Systems 157 (2006) 2128–2138.
101. D. W. Qiu, Automata theory based on quantum logic: Reversibilities and pushdown automata, Theoretical Computer Science 386 (2007) 38–56.
102. D. W. Qiu, F. C. Liu, Fuzzy discrete-event systems under fuzzy observability and a test algorithm, IEEE Transactions on Fuzzy Systems 17 (3) (2009), 578–589.
103. F. Ranzato, F. Tapparo, Generalizing the Paige-Tarjan algorithm by abstract interpretation, Information and Computation 206 (2008) 620–651.
104. M. Roggenbach, M. Majster-Cederbaum, Towards a unified view of bisimulation: a comparative study, Theoretical Computer Science 238 (2000) 81-130.
105. S. Roman, Lattices and Ordered Sets, Springer, New York, 2008.
106. D. Saha, An incremental bisimulation algorithm, In: V. Arvind, S. Prasad (eds.), FSTTCS 2007, Springer, Heidelberg, Lecture Notes in Computer Science 4855 (2007), 204–215.
107. E. Sanchez, Resolution of composite fuzzy relation equations, Information and Control 30 (1976) 38–48.
108. E. Sanchez, Solutions in composite fuzzy relation equations: application to medical diagnosis in Brouwerian logic, in: M. M. Gupta, G. N. Saridis, B. R. Gaines (Eds.), Fuzzy Automata and Decision Processes, North-Holland, Amsterdam, 1977, pp. 221–234.
109. E. Sanchez, Resolution of eigen fuzzy sets equations, Fuzzy Sets and Systems 1 (1978) 69–74.
110. D. Sangiorgi, On the origins of bisimulation and coinduction, ACM Transactions on Programming Languages and Systems 31 (4) (2009) 111–151.
111. E. S. Santos, Maximin automata, Information and Control 12 (1968) 367–377.
112. E. S. Santos, On reduction of maxmin machines, Journal of Mathematical Analysis and Applications 37 (1972) 677–686.
113. E. S. Santos, Fuzzy automata and languages, Information Sciences 10 (1976) 193–197.
114. S. S. Skiena, The Algorithm Design Manual, Springer, London, 2008.
115. A. Stamenković, M. Ćirić, Construction of fuzzy automata from fuzzy regular expressions, submitted to Fuzzy Sets and Systems.
116. A. Stamenković, M. Ćirić, J. Ignjatović, Reduction of fuzzy automata by means of fuzzy quasi-orders, submitted to Information Sciences.
117. D. D. Sun, Y. M. Li, W. W. Yang, Bisimulation relations for fuzzy finite automata, Fuzzy Systems and Mathematics 23 (2009) 92-100 (in Chinese).

118. S. S. Skiena, The Algorithm Design Manual, Springer, London, 2008.
119. S. Yu, Regular languages, in: G. Rozenberg, A. Salomaa (Eds.), Handbook of Formal Languages, vol. 1, Springer-Verlag, Berlin,Heidelberg, 1997, pp. 41110.35
120. R. van Glabbeek, B. Ploeger, Five determinization algorithms, in: O.H. Ibarra and B. Ravikumar (eds.), CIAA 2008, Lecture Notes in Computer Science 5148 (2008) 161–170.
121. R. van Glabbeek, B. Ploeger, Five determinization algorithms, CS-Report 08-14, Eindhoven University of Technology, 2008.
122. W. Wechler, The Concept of Fuzziness in Automata and Language Theory, Akademie-Verlag, Berlin, 1978.
123. W. G. Wee, On generalizations of adaptive algorithm and application of the fuzzy sets concept to pattern classification, Ph.D. Thesis, Purdue University, June 1967.
124. W. G. Wee, K. S. Fu, A formulation of fuzzy automata and its application as a model of learning systems, IEEE Transactions on Systems, Man and Cybernetics 5 (1969) 215–223.
125. L. H. Wu, D. W. Qiu, Automata theory based on complete residuated lattice-valued logic: Reduction and minimization, Fuzzy Sets and Systems 161 (2010) 1635–1656.
126. H. Xing, D. W. Qiu, Pumping lemma in context-free grammar theory based on complete residuated lattice-valued logic, Fuzzy Sets and Systems 160 (2009) 1141–1151.
127. H. Xing, D. W. Qiu, Automata theory based on complete residuated lattice-valued logic: A categorical approach, Fuzzy Sets and Systems 160 (2009) 2416–2428.
128. H. Xing, D. W. Qiu, F. C. Liu, Automata theory based on complete residuated lattice-valued logic: Pushdown automata, Fuzzy Sets and Systems 160 (2009) 1125–1140.
129. H. Xing, D. W. Qiu, F. C. Liu, Z. J. Fan, Equivalence in automata theory based on complete residuated lattice-valued logic, Fuzzy Sets and Systems 158 (2007) 1407–1422.
130. M. S. Ying, Automata theory based on quantum logic (I), International Journal of Theoretical Physics 39 (2000) 985–995.
131. M. S. Ying, Automata theory based on quantum logic (II), International Journal of Theoretical Physics 39 (2000) 2545–2557.
132. M. S. Ying, A theory of computation based on quantum logic (I), Theoretical Computer Science 344 (2005) 134–207.
133. S. Yu, Regular languages, in: G. Rozenberg, A. Salomaa (Eds.), Handbook of Formal Languages, vol. 1, Springer-Verlag, Berlin, Heidelberg, 1997, pp. 41–110.

Прилог 1.

# ИЗЈАВА О АУТОРСТВУ

Изјављујем да је докторска дисертација, под насловом

АЛГОРИТМИ ЗА ДЕТЕРМИНИЗАЦИЈУ ТЕЖИНСКИХ И ФАЗИ АУТОМАТЕ

- резултат сопственог истраживачког рада,
- да предложена дисертација, ни у целини, ни у деловима, није била предложена за добијање било које дипломе, према студијским програмима других високошколских установа,
- да су резултатикоректно наведени и
- да нисам кршио/ла ауторска права, нити злоупотребио/ла интелектуалну својину других лица.

У Нишу, _____

Аутор дисертације: Зорана Јанчић

_____

Потпис докторанда:

_____

Прилог 2.

ИЗЈАВА О ИСТОВЕТНОСТИ ШТАМПАНЕ И ЕЛЕКТРОНСКЕ ВЕРЗИЈЕ ДОКТОРСКЕ ДИСЕРТАЦИЈЕ

Име и презиме аутора: Зорана Јанчић

Студијски програм: Информатика

Наслов рада: АЛГОРИТМИ ЗА ДЕТЕРМИНИЗАЦИЈУ ТЕЖИНСКИХ И ФАЗИ АУТОМАТЕ

Ментор: Јелена Игњатовић

Изјављујем да је штампана верзија моје докторске дисертације истоветна електронској верзији, коју сам предао/ла за уношење у Дигитални репозиторијум Универзитета у Нишу.

Дозвољавам да се објаве моји лични подаци, који су у вези са добијањем академског звања доктора наука, као што су име и презиме, година и место рођења и датум одбране рада, и то у каталогу Библиотеке, Дигиталном репозиторијуму Универзитета у Нишу, као и у публикацијама Универзитета у Нишу.

У Нишу, _____

Аутор дисертације:

Зорана Јанчић

Потпис докторанда:

_____

Прилог 3.

ИЗЈАВА О КОРИШЋЕЊУ

Овлашћујем Универзитетску библиотеку „Никола Тесла“ да, у Дигитални репозиторијум Универзитета у Нишу, унесе моју докторску дисертацију, под насловом:

АЛГОРИТМИ ЗА ДЕТЕРМИНИЗАЦИЈУ ТЕЖИНСКИХ И ФАЗИ АУТОМАТА

која је моје ауторско дело.

Дисертацију са свим прилозима предао/ла сам у електронском формату, погодном за трајно архивирање.

Моју докторску дисертацију, унету у Дигитални репозиторијум Универзитета у Нишу, могу користити сви који поштују одредбе садржане у одабраном типу лиценце Креативне заједнице (Creative Commons), за коју сам се одлучио/ла.

1. Ауторство
2. Ауторство – некомерцијално
3. Ауторство – некомерцијално – без прераде
4. Ауторство – некомерцијално – делити под истим условима
5. Ауторство – без прераде
6. Ауторство – делити под истим условима

(Молимо да подвучете само једну од шест понуђених лиценци; кратак опис лиценци је у наставку текста).

У Нишу, _____

Аутор дисертације: Зорана Јанчић

_____

Потпис докторанда

_____

# ПРИРОДНО - МАТЕМАТИЧКИ ФАКУЛТЕТ
## НИШ

# КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА

| | |
|---|---|
| Редни број, **РБР**: | |
| Идентификациони број, **ИБР**: | |
| Тип документације, **ТД**: | монографска |
| Тип записа, **ТЗ**: | текстуални / графички |
| Врста рада, **ВР**: | докторска дисертација |
| Аутор, **АУ**: | Зорана З. Јанчић |
| Ментор, **МН**: | Јелена Игњатовић |
| Наслов рада, **НР**: | Алгоритми за детерминизацију тежинских и фази аутомата |
| Језик публикације, **ЈП**: | енглески |
| Језик извода, **ЈИ**: | српски |
| Земља публиковања, **ЗП**: | Србија |
| Уже географско подручје, **УГП**: | Србија |
| Година, **ГО**: | 2014. |
| Издавач, **ИЗ**: | ауторски репринт |
| Место и адреса, **МА**: | Ниш, Вишеградска 33. |
| Физички опис рада, **ФО**: | 144 стр., граф. прикази |
| Научна област, **НО**: | Рачунарске науке |
| Научна дисциплина, **НД**: | Теорија изречунавања |
| Предметна одредница/Кључне речи, **ПО**: | фази аутомати и језици, детерминизација |
| **УДК** | 519.71, 519.713, 519.76 |
| Чува се, **ЧУ**: | библиотека |
| Важна напомена, **ВН**: | |
| Извод, **ИЗ**: | Главни задатак дисертације је развијање нових метода и алгоритама за детерминизацију тежинских и фази аутомата. Под детрминизацијом тежинских односно фази аутомата би се сматрао поступак конверзије ових аутомата у одговарајуће детерминистичке тежинске односно фази аутомате, који распознају исти језик као полазни аутомати. Неки од алгоритама који се изучавају представљају истовремено алгоритме за канонизацију, другим речима врше детерминизацију и као производ дају минимални детерминистички тежински односно фази аутомат еквивалентан полазном аутомату. Даље, је разматрана употреба десно и лево инваријантних фази-квази уређења при детерминизацији фази аутомата. Тачније за фази аутомата и десно (лево) инваријантно фази квази уређење на њему конструисан је еквивалентан детерминистички фази аутомат. |
| Датум прихватања теме, **ДП**: | 14.11.2012. |
| Датум одбране, **ДО**: | |
| Чланови комисије, **КО**: Председник: | |
| Члан: | |
| Члан, ментор: | |

# ПРИРОДНО - МАТЕМАТИЧКИ ФАКУЛТЕТ
## НИШ

# KEY WORDS DOCUMENTATION

| | |
|---|---|
| Accession number, **ANO**: | |
| Identification number, **INO**: | |
| Document type, **DT**: | **monograph** |
| Type of record, **TR**: | **textual / graphic** |
| Contents code, **CC**: | **doctoral dissertation** |
| Author, **AU**: | **Zorana Z. Jančić** |
| Mentor, **MN**: | **Jelena Ignjatović** |
| Title, **TI**: | Algorithms for determinization of weighted and fuzzy automata |
| Language of text, **LT**: | **English** |
| Language of abstract, **LA**: | **Serbian** |
| Country of publication, **CP**: | **Serbia** |
| Locality of publication, **LP**: | **Serbia** |
| Publication year, **PY**: | **2014** |
| Publisher, **PB**: | **author's reprint** |
| Publication place, **PP**: | **Niš, Višegradska 33.** |
| Physical description, **PD**: | **144 p. ; graphic representations** |
| Scientific field, **SF**: | **Computer Science** |
| Scientific discipline, **SD**: | **Theory of computing** |
| Subject/Key words, **S/KW**: | **fuzzy automata and languages, determinization** |
| **UC** | 519.71, 519.713, 519.76 |
| Holding data, **HD**: | **library** |
| Note, **N**: | |
| Abstract, **AB**: | The main objective of the thesis is to the develop of methods and algorithms for detereminization of weighted and fuzzy automata. The determinization of weighted (fuzzy) automata is considered here to be a procedure of its conversion into the corresponding deterministic weighted (fuzzy) automata, which recognize the same language. Some of the algorithms that studied here are canonization algorithms. Further, we consider the use of the right and left invariant fuzzy-quasi order relations in determinization of fuzzy automat. Specifically, given a fuzzy automaton and the right invariant fuzzy-quasi order relation on it we give an algorithm for constraction of an equivalent deterministic fuzzy automaton. |
| Accepted by the Scientific Board on, | 14.11.2012. |
| Defended on, **DE**: | |
| Defended Board, President: | |
| Member: | |
| Member, | |