Volume 13 Number 1

ACTA CYBERNETICA

Szeged, 1997

ACTA CYBERNETICA

Information for authors. Acta Cybernetica publishes only original papers in the field of Computer Science. Contributions are accepted for review with the understanding that the same work has not been published elsewhere.

Manuscripts must be in English and should be sent in triplicate to any of the Editors. On the first page, the *title* of the paper, the *name(s)* and *affiliation(s)*, together with the *mailing* and *electronic address(es)* of the author(s) must appear. An *abstract* summarizing the results of the paper is also required. References should be listed in alphabetical order at the end of the paper in the form which can be seen in any article already published in the journal. Manuscripts are expected to be made with a great care. If typewritten, they should be typed double-spaced on one side of each sheet. Authors are encouraged to use any available dialect of TEX.

After acceptance, the authors will be asked to send the manuscript's source TEX file, if any, on a diskette to the Managing Editor. Having the TEX file of the paper can speed up the process of the publication considerably. Authors of accepted contributions may be asked to send the original drawings or computer outputs of figures appearing in the paper. In order to make a photographic reproduction possible, drawings of such figures should be on separate sheets, in India ink, and carefully lettered.

There are no page charges. Fifty reprints are supplied for each article published.

Publication information. Acta Cybernetica (ISSN 0324–721X) is published by the Department of Informatics of the József Attila University, Szeged, Hungary. Each volume consists of four issues, two issues are published in a calendar year. For 1997 Numbers 1–2 of Volume 13 are scheduled. Subscription prices are available upon request from the publisher. Issues are sent normally by surface mail except to overseas countries where air delivery is ensured. Claims for missing issues are accepted within six months of our publication date. Please address all requests for subscription information to: Department of Informatics, József Attila University, H-6701 Szeged, P.O.Box 652, Hungary. Tel.: (36)-(62)-311-184, Fax:(36)-(62)-312-292.

URL access. All these information and the contents of the last some issues are available in the Acta Cybernetica home page at http://www.inf.u-szeged.hu/local/acta.

H. Bunke
Universität Bern
Institut für Informatik und
angewandte Mathematik
Längass strasse 51., CH-3012 Bern
Switzerland

Courcelle
Universitè Bordeaux-1
LaBRI, 351 Cours de la Libèration
33405 TALENCE Cedex
France

J. Demetrovics
MTA SZTAKI
Budapest, P.O.Box 63
H-1502 Hungary

B. Dömölki
IQSOFT
Budapest, Teleki Blanka u. 15-17.
H-1142 Hungary

J. Engelfriet
Leiden University
Computer Science Department
P.O. Box 9512, 2300 RA LEIDEN
The Netherland

Z. Ésik
A. József University
Department of Foundations of
Computer Science
Szeged, Aradi vértanúk tere 1.
H-6720 Hungary

L. Lovász
Eötvös Loránd University
Budapest Múzeum krt. 6-8.
H-1088 Hungary

G. Paun
Institute of Mathematics
Romanian Academy
P.O.Box 1-764, RO-70700
Bucuresti, Romania

A. Prékopa
Eötvös Loránd University
Budapest, Múzeum krt. 6-8.
H-1088 Hungary

A. Salomaa
University of Turku
Department of Mathematics
SF-20500 Turku 50, Finland

L. Varga
Eötvös Loránd University
Budapest, Múzeum krt. 6-8.
H-1088 Hungary

H. Vogler
Dresden University of Technology
Faculty of Computer Science
Foundations of Programming
D-01062 Dresden, Germany

G. Wöginger
Technische Universität Graz
Institut für Mathematik (501B)
Steyrergasse 30
A-8010 Graz, Österreich

Acta Cybernetica **13** (1997) 1-21.

Regular expression star-freeness is PSPACE-complete

László Bernátsky *†

Abstract

It is proved that the problem of deciding if a regular expression denotes a star-free language is **PSPACE**-complete. The paper also includes a new proof of the **PSPACE**-completeness of the finite automaton aperiodicity problem.

1 Introduction

Star-free languages form an important subclass of regular languages: they are the ones that can be obtained from the singleton languages by a finite number of applications of the operations of union, complement and product. By Schützenberger's famous theorem [8], a regular language is star-free if and only if its syntactic monoid is aperiodic, or equivalently, if it is recognized by an aperiodic DFA. Moreover, a language is star-free if and only if it can be defined by a first-order formula of a suitable formal language, see [10]. In his 1985 paper [9], Jacques Stern proved that the problem of deciding whether a DFA is aperiodic is **Co-NP**-hard and belongs to **PSPACE**. A few years later Sang Cho and Dung T. Huynh strengthened Stern's result by showing that this problem is in fact **PSPACE**-complete, see [3]. Not knowing about their work I proved the same result while Zoltán Ésik and I were working on the description of the free Conway theories, see [1]. The present paper contains a slightly modified version of my original proof, which rests on the same basic idea as the proof of Cho and Huynh, but uses a different construction, see Construction 4.1. This different construction makes it easy to extend the proof to regular expressions.

2 Definitions and preliminary facts

2.1 Sets and relations

The set of nonnegative integers is denoted $\mathbf{N}$, and ω stands for the set of positive integers. For $n \in \mathbf{N}$, $[n]$ denotes the set $\{1, \ldots, n\}$, so that $[0]$ is another name for the empty set $\emptyset$.

*Department of Computer Science, Attila József University, 6720 Szeged, Hungary, E-mail: benny@inf.u-szeged.hu

†Supported in part by grant no. T7383 of the National Foundation of Scientific Research of Hungary

The power-set $\mathrm{P}(S)$ of a set S consists of all subsets of S, and the direct product $A \times B$ of two sets A and B consists of all pairs (a, b) with $a \in A$ and $b \in B$. A binary relation from A to B is just a subset of $A \times B$, so that $\mathrm{P}(A \times B)$ is the set of all binary relations from A to B. The composite of two binary relations $\rho \subseteq A \times B$ and $\rho' \subseteq B \times C$ is the relation

$$\rho \circ \rho' \;=\; \{(a,c) \mid \exists b \in B \;\; (a,b) \in \rho \wedge (b,c) \in \rho'\} \;\subseteq\; A \times C.$$

We use the infix notation $a\rho b$ instead of $(a, b) \in \rho$. Suppose that A' is subset of A, and B' is a subset of B. We write $A'\rho B'$ if there exist $a \in A'$ and $b \in B'$ with $a\rho b$. The image of A' under ρ is denoted $A'\rho$, i.e.,

$$A'\rho \;=\; \{b \in B \mid \exists a \in A' \;\; a\rho b\}.$$

When $A' = \{a\}$ is a singleton, we write $a\rho$ instead of $A'\rho$.

2.2 Words and languages

Suppose that A and B are **alphabets**, i.e., nonempty finite sets. We denote by A^* the set of all finite words over A including the empty word ϵ, while A^+ stands for $A^* \setminus \{\epsilon\}$. The set A^ω is the collection of all infinite words over A. The length of a finite word $u \in A^*$ is denoted $|u|$, and the ith letter of a finite or infinite word $w \in A^* \cup A^\omega$ is denoted w_i. Thus, any finite word $u \in A^*$ can be written as $u_1 u_2 \ldots u_{|u|}$, where each u_i is an element of A. A word $v \in A^*$ is called a **prefix** of a word $u \in A^* \cup A^\omega$ if $u = vw$, for some $w \in A^* \cup A^\omega$. A function $\varphi : A^* \to B^*$ is called a **homomorphism** if $\varphi(uv) = \varphi(u)\varphi(v)$, for all words $u, v \in A^*$. Note that each homomorphism $A^* \to B^*$ is totally determined by its restriction to A.

For the reader's convenience we restate the theorem of Schützenberger.

THEOREM 2.1 (Schützenberger [8]) *A regular language $L \subseteq \Sigma^*$ is star-free if and only if there exists some integer $k \geq 0$ such that*

$$uv^k w \in L \iff uv^{k+1} w \in L,$$

for all words $u, v, w \in \Sigma^*$.

A proof of the following lemma is given in the appendix.

LEMMA 2.1 *Suppose that $\Sigma = \{\sigma_1, \ldots, \sigma_n\}$ is an alphabet and $l = \lceil \log_2(n+1) \rceil$. Then there exists an injective homomorphism $\psi : \Sigma^* \to \{0,1\}^*$ satisfying the following conditions.*

- *The ψ-image of each letter $\sigma \in \Sigma$ is a word of length $2l$ beginning with a sequence of l zeros and containing the letter 1. In other words,*

$$\psi(\Sigma) \;\subseteq\; 0^l\{0,1\}^l \setminus \{0^{2l}\}. \tag{1}$$

- *For all words* $u, v, w \in \{0,1\}^*$

$$uv^{2l}w \in \psi(\Sigma^*) \implies 2l \text{ divides } |v|. \tag{2}$$

- *For all regular languages* $L \subseteq \Sigma^*$

$$L \text{ is star-free} \iff \psi(L) \text{ is star-free.} \tag{3}$$

2.3 Regular expressions

Let $\mathcal{R}$ be the ranked alphabet consisting of the constant symbol $\emptyset$, unary symbols $^*, \sim$ and binary symbols $\cdot, \cup, \cap$. Suppose that Σ is an alphabet such that $\Sigma \cap \mathcal{R} = \emptyset$. For any subset $\mathcal{R}'$ of $\mathcal{R}$, the set $\Sigma \cup \mathcal{R}'$ can be considered as a ranked alphabet in which the elements of Σ have rank 0, and the elements of $\mathcal{R}'$ have the same rank as in $\mathcal{R}$. An **$\mathcal{R}'$-type regular expression over** Σ is a ground $(\Sigma \cup \mathcal{R}')$-term, i.e., a term over the ranked alphabet $\Sigma \cup \mathcal{R}'$ containing no variable symbols. A $\{\emptyset, \cdot, \cup, ^*\}$-type regular expression is simply called a **regular expression**, and an $(\mathcal{R} \setminus \{^*\})$-type regular expression is also called a **star-free regular expression**.

As for the syntactical conventions, we use infix notation for the binary operations $\cup, \cap$ and $\cdot$, postfix notation for *, and we write $\overline{a}$ instead of $\sim a$. The operation symbol $\cdot$ is usually omitted. If $\Sigma' = \{\sigma_1, \ldots, \sigma_n\}$ is a subset of Σ, we simply write Σ' instead of $\sigma_1 \cup \cdots \cup \sigma_n$.

The language $L(E) \subseteq \Sigma^*$ denoted by an $\mathcal{R}$-type regular expression E over Σ is defined in the usual way, see [7]. Note that a regular language $L \subseteq \Sigma^*$ is star-free if and only if it is denoted by some star-free regular expression over Σ.

We recall from [7] that the **star-height** $\mathrm{sh}(E)$ of a regular expression E is defined by

$$\begin{aligned}
\mathrm{sh}(\sigma) &= 0 \\
\mathrm{sh}(\emptyset) &= 0 \\
\mathrm{sh}(E \cup F) &= \max\{\mathrm{sh}(E), \mathrm{sh}(F)\} \\
\mathrm{sh}(E \cdot F) &= \max\{\mathrm{sh}(E), \mathrm{sh}(F)\} \\
\mathrm{sh}(E^*) &= 1 + \mathrm{sh}(E),
\end{aligned}$$

for all letters $\sigma \in \Sigma$ and regular expressions E, F over Σ.

2.4 Finite automata

Most of our automata-theoretical notations and definitions are adopted from [4].

A **(nondeterministic) finite automaton** (NFA) is represented as a 5-tuple $\mathcal{A} = (Q, \Sigma, \tau, I, F)$, where

- Q is the finite set of states,
- Σ is the input alphabet,

- $\tau : \Sigma \to \mathrm{P}(Q \times Q)$ is the transition function,
- $I \subseteq Q$ is the set of initial states,
- $F \subseteq Q$ is the set of final states.

Note that for each input symbol $\sigma \in \Sigma$, $\tau(\sigma)$ is a binary relation on Q, called the **relation induced by σ in the automaton $\mathcal{A}$**. We prefer the notation $\sigma_{\mathcal{A}}$ to $\tau(\sigma)$. When $u \in \Sigma^*$ is an input word, $u_{\mathcal{A}}$ denotes the **relation induced by u in $\mathcal{A}$**, defined by

$$u_{\mathcal{A}} \quad := \quad \tau(u_1) \circ \cdots \circ \tau(u_{|u|}).$$

Note that $\epsilon_{\mathcal{A}}$ is the identity relation.

The automaton $\mathcal{A}$ can be visualized as a directed graph with vertices Q, and edges labeled by input symbols in Σ. Motivated by this point of view, we shall sometimes denote the relation $u_{\mathcal{A}}$ by $\xrightarrow{u}_{\mathcal{A}}$. Then $q \xrightarrow{u}_{\mathcal{A}} q'$ means that there is a directed u-labeled path from vertex q to vertex q'.

The language $L(\mathcal{A})$ recognized by $\mathcal{A}$ consists of those words $u \in \Sigma^*$ for which there exists a u-labeled path from some initial state to a final state, formally

$$L(\mathcal{A}) \quad = \quad \{u \in \Sigma^* \mid I \xrightarrow{u}_{\mathcal{A}} F\}.$$

When $\mathcal{A}$ is understood, we sometimes omit the subscript in $\xrightarrow{u}_{\mathcal{A}}$ and $u_{\mathcal{A}}$.

We call $\mathcal{A}$ a **deterministic finite automaton** (DFA) if it has at most one initial state, and each relation $\sigma_{\mathcal{A}}$ ($\sigma \in \Sigma$) is a *partial* function $Q \to Q$. A deterministic automaton is called **complete** if it has a unique initial state, and each of its input symbols induces a total function.

The automaton $\mathcal{A}$ is called a **reset automaton** if it has at most one initial state and each input symbol $\sigma \in \Sigma$ induces either the identity function or a partial constant function $Q \to Q$.

A state q of $\mathcal{A}$ is called **accessible** (respectively, **coaccessible**) if there exists some input word $u \in \Sigma^*$ with $I \xrightarrow{u}_{\mathcal{A}} \{q\}$ (respectively, $\{q\} \xrightarrow{u}_{\mathcal{A}} F$). Note that each initial state is accessible and each final state is coaccessible. A **biaccessible** state is one which is both accessible and coaccessible. Two states $q, q' \in Q$ are called **equivalent**, denoted $q \approx_{\mathcal{A}} q'$, if

$$\{q\} \xrightarrow{u}_{\mathcal{A}} F \quad \Longleftrightarrow \quad \{q'\} \xrightarrow{u}_{\mathcal{A}} F,$$

for all input words $u \in \Sigma^*$. Suppose that $\mathcal{A}$ is a DFA. Then $\mathcal{A}$ is called

minimal if all of its states are biaccessible, and it has no different equivalent states,

aperiodic if there exists an integer $k \geq 0$ such that $(u^k)_{\mathcal{A}} = (u^{k+1})_{\mathcal{A}}$, for all $u \in \Sigma^*$.

Observe that if $\mathcal{A}$ is a reset automaton then $(u^2)_{\mathcal{A}} = (u^3)_{\mathcal{A}}$, and if $\mathcal{A}$ is a complete reset automaton then $u_{\mathcal{A}} = (u^2)_{\mathcal{A}}$, for all words $u \in \Sigma^*$.

REMARK 2.1 *It is well known (see [4]) that a deterministic automaton* $\mathcal{A} = (Q, \Sigma, \tau, I, F)$ *is aperiodic if and only if it satisfies the implication*

$$q \xrightarrow{u^k}_{\mathcal{A}} q \implies q \xrightarrow{u}_{\mathcal{A}} q,$$

for all states $q \in Q$, *input words* $u \in \Sigma^+$ *and integers* $k \geq 2$.

Suppose that $n \geq 1$, and $\mathcal{A}_i = (Q_i, \Sigma, \tau_i, I_i, F_i)$ is an NFA, for each $i \in [n]$. Then the **product** of the A_i's is the NFA

$$\prod_{i\in[n]} \mathcal{A}_i \;=\; (\prod_{i\in[n]} Q_i,\ \Sigma,\ \tau,\ \prod_{i\in[n]} I_i,\ \prod_{i\in[n]} F_i),$$

where

$$\tau(\sigma) \;=\; \{((q_1,\ldots,q_n),(r_1,\ldots,r_n)) \mid \forall i \in [n]\ (q_i, r_i) \in \tau_i(\sigma)\},$$

for all $\sigma \in \Sigma$. It is easy to see that

$$L(\prod_{i\in[n]} \mathcal{A}_i) \;=\; \bigcap_{i\in[n]} L(\mathcal{A}_i).$$

2.5 Turing machines

A **deterministic Turing machine** (DTM) with a single one-way infinite tape is a system $\mathcal{M} = (Q, \Gamma, \Sigma, \delta, q_0, q_f)$, where

- Q is the finite set of states,
- Γ is the tape alphabet containing the special "blank" symbol $\flat$,
- $\Sigma \subseteq \Gamma$ is the input alphabet, $\flat \notin \Sigma$,
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{-1, 0, 1\}$ is the partial transition function,
- $q_0 \in Q$ is the initial state,
- $q_f \in Q$ is the final state.

We say the machine $\mathcal{M}$ is in the **configuration** (q, i, u) for a state $q \in Q$, integer $i \in \omega$ and infinite word $u \in \Gamma^\omega$ if in state q it scans the ith tape cell and the content of the tape is u. We define a binary relation $\vdash_{\mathcal{M}}$ on the set $Q \times \omega \times \Gamma^\omega$ of configurations by

$$(q, i, u) \vdash_{\mathcal{M}} (r, j, v) \iff \delta(q, u_i) = (r, v_i, j - i) \wedge \forall t \in \omega\ (t \neq i \implies v_t = u_t).$$

Note that $\vdash_{\mathcal{M}}$ is a partial function. The machine $\mathcal{M}$ **accepts** an input word $u \in \Sigma^*$ if

$$(q_0, 1, u\flat^\omega) \vdash^*_{\mathcal{M}} (q_f, 1, \flat^\omega),$$

otherwise $\mathcal{M}$ **rejects** u. The **language** $L(\mathcal{M}) \subseteq \Sigma^*$ **recognized by** $\mathcal{M}$ consists of those words $u \in \Sigma^*$ which are accepted by $\mathcal{M}$. Thus, for each word $u \in L(\mathcal{M})$, there exists a shortest sequence $(q_1, i_1, w_1), \ldots, (q_k, i_k, w_k)$ of configurations such that

$$\begin{array}{rcll} (q_1, i_1, w_1) & = & (q_0, 1, ub^\omega), & \\ (q_k, i_k, w_k) & = & (q_f, 1, b^\omega), & \text{and} \\ (q_t, i_t, w_t) & \vdash_{\mathcal{M}} & (q_{t+1}, i_{t+1}, w_{t+1}), & \end{array}$$

for all $t \in [k-1]$. Then we define

$$\mathrm{SPACE}_{\mathcal{M}}(u) \;:=\; \max_{t \in [k]} i_t.$$

Suppose that $S : \mathbf{N} \to \mathbf{N}$ is a (space constructible) function. The machine $\mathcal{M}$ is said to have **space complexity** S if $\mathrm{SPACE}_{\mathcal{M}}(u) \leq S(|u|)$, for all words $u \in L(\mathcal{M})$. The language class **PSPACE** consists of those languages which are recognized by some Turing machine $\mathcal{M}$ having space-complexity p, for some polynomial function $p : \mathbf{N} \to \mathbf{N}$.

We assume the reader is familiar with the concept of logspace-reducibility (see [2], for example).

Suppose L and L' are languages. In this paper, $L \leq_{log} L'$ stands for "L is logspace-reducible to L'". The language L is called **PSPACE-hard** with respect to logspace-reductions, written **PSPACE** $\leq_{log} L$, if every language in **PSPACE** is logspace-reducible to L. Lastly, L is called **PSPACE-complete** with respect to logspace-reductions if $L \in$ **PSPACE** and **PSPACE** $\leq_{log} L$.

3 Problems

We are interested in the computational complexity of the following decision problems:

1. The automata intersection problem (**AIP**):

   INPUT: A sequence $\mathcal{A}_1, \ldots, \mathcal{A}_n$ $(n \geq 2)$ of nondeterministic finite automata with a common input alphabet.

   QUESTION: Does $\bigcap_{i \in [n]} L(\mathcal{A}_i) \neq \emptyset$ hold?

2. A restricted version of the automata intersection problem ($\mathbf{AIP}_R$):

   INPUT: A sequence $\mathcal{A}_1, \ldots, \mathcal{A}_n$ $(n \geq 2)$ of minimal reset automata with a common input alphabet.

   QUESTION: Does $\bigcap_{i \in [n]} L(\mathcal{A}_i) \neq \emptyset$ hold?

3. Automaton star-freeness (**ASF**):

INPUT: A nondeterministic finite automaton $\mathcal{A}$.

QUESTION: Does $\mathcal{A}$ recognize a star-free language?

4. A restricted version of automaton star-freeness ($\mathbf{ASF}_R$):

   INPUT: A minimal DFA $\mathcal{A}$ with input alphabet {0,1}.

   QUESTION: Does $\mathcal{A}$ recognize a star-free language?

5. Regular expression star-freeness (**RSF**):

   INPUT: A regular expression E.

   QUESTION: Does E denote a star-free language?

6. A restricted version of regular expression star-freeness ($\mathbf{RSF}_R$):

   INPUT: A regular expression E of star-height 2 over the alphabet $\{0, 1\}$.

   QUESTION: Does E denote a star-free language?

Assuming some efficient encoding of automata and regular expressions (see [5]) with words over a fixed finite alphabet, all these problems can be considered as languages. We are going to prove

PROPOSITION 3.1 *The problems* **AIP**, $\mathbf{AIP}_R$, **ASF**, $\mathbf{ASF}_R$, **RSF** *and* $\mathbf{RSF}_R$ *are* **PSPACE***-complete with respect to logspace reductions.*

4 Constructions

In this section we present the constructions of automata and regular expressions which are needed to show that the restricted problems $\mathbf{AIP}_R$, $\mathbf{ASF}_R$ and $\mathbf{RSF}_R$ are **PSPACE**-hard. The first construction shows how can one replace a deterministic Turing machine with a sequence of reset automata.

CONSTRUCTION 4.1 **Input:** A polynomial function $p : \mathbf{N} \to \mathbf{N}$, a DTM $\mathcal{M} = (Q, \Gamma, \Sigma, \delta, q_0, q_f)$ of space-complexity p, and an input word $u \in \Sigma^n$, $n \geq 0$.
Output: A sequence $\mathcal{S}, \mathcal{P}, \mathcal{A}_1, \ldots, \mathcal{A}_m$ of reset automata, where $m = \max\{p(n), 1\}$, and

$$u \in L(\mathcal{M}) \iff L(\mathcal{S}) \cap L(\mathcal{P}) \cap \bigcap_{i \in [m]} L(A_i) \neq \emptyset. \tag{4}$$

Description: Let

$$\begin{aligned} \mathcal{S} &= (Q, A, \tau_{\mathcal{S}}, \{q_0\}, \{q_f\}) \\ \mathcal{P} &= ([m], A, \tau_{\mathcal{P}}, \{1\}, \{1\}), \end{aligned}$$

and for each $i \in [m]$,

$$\mathcal{A}_i = (\Gamma, A, \tau_i, \{(ub^\omega)_i\}, \{b\}),$$

where

$$A \;=\; \{\langle q,k,\gamma\rangle \mid q \in Q,\ k \in [m],\ \gamma \in \Gamma\}$$

and the transition functions $\tau_{\mathcal{S}}, \tau_{\mathcal{P}}, \tau_1, \ldots, \tau_m$ are defined as follows.

Suppose that $a = \langle q,k,\gamma\rangle$ is an element of A. If $\delta(q,\gamma)$ is undefined then

$$\tau_{\mathcal{S}}(a) = \tau_{\mathcal{P}}(a) = \tau_1(a) = \cdots = \tau_m(a) = \emptyset,$$

and if $\delta(q,\gamma)$ is defined, say $\delta(q,\gamma) = (r,\gamma',t)$, then

$$\begin{aligned}
\tau_{\mathcal{S}}(a) &= \{(q,r)\} \\
\tau_{\mathcal{P}}(a) &= \begin{cases} \{(k,k+t)\} & \text{if } k+t \in [m], \\ \emptyset & \text{if } k+t \notin [m], \end{cases} \\
\tau_i(a) &= \begin{cases} \{(\gamma,\gamma')\} & \text{if } k = i \\ \{(\sigma,\sigma) \mid \sigma \in \Gamma\} & \text{if } k \neq i. \end{cases}
\end{aligned}$$

Proof. The intuition is that the automata $\mathcal{S}, \mathcal{P}, \mathcal{A}_1, \ldots, \mathcal{A}_m$ together "simulate" the computation of $\mathcal{M}$ on the input word u, such that $\mathcal{S}$ knows the current state of $\mathcal{M}$, $\mathcal{P}$ knows the position of the read-write head, and each $\mathcal{A}_i$ ($i \in [m]$) knows the content of the ith tape-cell. An input symbol $\langle q,k,\gamma\rangle \in A$ corresponds to the statement "the current state of $\mathcal{M}$ is q, the position of the read-write head is k, and the content of the kth tape-cell is γ".

It is easy to see that each one of $\mathcal{S}, \mathcal{P}, \mathcal{A}_1, \ldots, \mathcal{A}_m$ is a reset automaton. (In fact they are even more restricted: for all input symbols $a \in A$, the relation induced by a in each one of the automata $\mathcal{S}, \mathcal{P}, \mathcal{A}_1, \ldots, \mathcal{A}_m$ is either empty, or a singleton, or the identity function.)

Consider the product automaton

$$\mathcal{A} \;=\; \mathcal{S} \times \mathcal{P} \times \prod_{i \in [m]} \mathcal{A}_i.$$

We know

$$L(\mathcal{A}) \;=\; L(\mathcal{S}) \cap L(\mathcal{P}) \cap \bigcap_{i \in [m]} L(\mathcal{A}_i).$$

Observe that for all $q, r \in Q$, $v, w \in \Gamma^m$, $j, k \in [m]$, and $a \in A$

$$\begin{aligned}
&(q, j, v_1, \ldots, v_m) \xrightarrow{a}_{\mathcal{A}} (r, k, w_1, \ldots, w_m) \quad \Longleftrightarrow \\
&\qquad a = \langle q, j, v_j\rangle \ \wedge\ \delta(q, v_j) = (r, w_j, k-j) \ \wedge\ \forall t \in [m]\,(t \neq j \Rightarrow w_t = v_t),
\end{aligned}$$

and thus

$$(q, j, vb^\omega) \vdash_{\mathcal{M}} (r, k, wb^\omega) \quad \Longleftrightarrow \quad \exists a \in A\ (q, j, v_1, \ldots, v_m) \xrightarrow{a}_{\mathcal{A}} (r, k, w_1, \ldots, w_m).$$

It follows that

$$\begin{aligned} u \in L(\mathcal{M}) &\iff (q_0, 1, ub^\omega) \vdash^*_{\mathcal{M}} (q_f, 1, b^\omega) \\ &\iff \exists v \in A^* \ (q_0, 1, (ub^\omega)_1, \ldots, (ub^\omega)_m) \xrightarrow{v}_{\mathcal{A}} (q_f, 1, b, \ldots, b) \\ &\iff L(\mathcal{A}) \neq \emptyset. \end{aligned}$$

□

Although the automata $\mathcal{S}, \mathcal{P}, \mathcal{A}_1, \ldots, \mathcal{A}_m$ constructed above have a very simple structure, they are not always minimal. In the next construction we show an easy way of modifying these automata so that they become minimal. Note that the standard procedure of automata minimization is not suitable for our purposes since it requires linear space.

CONSTRUCTION 4.2 **Input:** A sequence $\mathcal{A}_1, \ldots, \mathcal{A}_n$ $(n \geq 2)$ of reset automata of the form $\mathcal{A}_i = (Q_i, \Sigma, \tau_i, \{s_i\}, \{f_i\})$.
Output: A sequence $\mathcal{B}_1, \ldots, \mathcal{B}_n$ of minimal reset automata such that

$$\bigcap_{i \in [n]} L(\mathcal{A}_i) \quad = \quad \bigcap_{i \in [n]} L(\mathcal{B}_i). \tag{5}$$

Description: For each $i \in [n]$ let

$$\mathcal{B}_i \quad = \quad (Q_i, \Sigma \cup \Sigma', \tau_i', \{s_i\}, \{f_i\}),$$

where

$$\begin{aligned} \Sigma' &= \{\langle q, j\rangle \mid q \in Q_j, \ j \in [n]\}, \\ \tau_i'(\sigma) &= \tau_i(\sigma), \\ \tau_i'(\langle q, j\rangle) &= \begin{cases} \{(p, q) \mid p \in Q_i, \ p \neq q\} & \text{if } j = i, \\ \emptyset & \text{if } j \neq i, \end{cases} \end{aligned}$$

for all $\sigma \in \Sigma$, $\langle q, j\rangle \in \Sigma'$.

Proof. For each $j \in [n]$ let Σ_j' denote the set $\{\langle q, j\rangle \mid q \in Q_j\}$. Consider the automaton $\mathcal{B}_i$ for some $i \in [n]$. It is obvious that $\mathcal{B}_i$ is a reset automaton. Since the elements of $\Sigma' \setminus \Sigma_i'$ induce the empty relation in $\mathcal{B}_i$,

$$L(\mathcal{B}_i) \quad \subseteq \quad (\Sigma \cup \Sigma_i')^*.$$

Moreover, since each input symbol $\sigma \in \Sigma$ induces the same relation in $\mathcal{B}_i$ as in $\mathcal{A}_i$,

$$L(\mathcal{B}_i) \cap \Sigma^* \quad = \quad L(\mathcal{A}_i).$$

These two observations and $n \geq 2$ imply (5). Lastly, for all states $p, q \in Q_i$ we have

$$\begin{aligned} q \neq s_i &\implies s_i \xrightarrow{\langle q, i\rangle} q, \\ q \neq f_i &\implies q \xrightarrow{\langle f_i, i\rangle} f_i, \\ p \neq q \ \wedge \ q \neq f_i &\implies p\langle q, i\rangle\langle f_i, i\rangle = \{f_i\} \ \wedge \ q\langle q, i\rangle\langle f_i, i\rangle = \emptyset, \end{aligned}$$

showing $\mathcal{B}_i$ is minimal. □

The next construction shows that for each reset automaton $\mathcal{A}$ there exists a "short" regular expression denoting the complement of the language recognized by $\mathcal{A}$. This fact plays a key role in proving that the problem $\mathbf{RSF}_R$ is **PSPACE**-hard.

CONSTRUCTION 4.3 **Input:** A reset automaton

$$\mathcal{A} = (Q, \Sigma, \tau, I, F).$$

Output: A regular expression E over the alphabet Σ such that

$$L(E) = \overline{L(\mathcal{A})}. \tag{6}$$

Description: If $I = \emptyset$ then (6) holds for the regular expression $E = \Sigma^*$. From now on we assume that $\mathcal{A}$ has an initial state q_0. Let

$$\begin{aligned} X_q &= \{\sigma \in \Sigma \mid Q\sigma_{\mathcal{A}} = \{q\}\} \\ Y_q &= \{\sigma \in \Sigma \mid q\sigma_{\mathcal{A}} = \{q\}\} \\ Z_q &= \{\sigma \in \Sigma \mid q\sigma_{\mathcal{A}} = \emptyset\}, \end{aligned}$$

for all $q \in Q$. Using these subsets of Σ we define the regular expressions

$$E_q = \begin{cases} \Sigma^* X_q Y_q^* & \text{if } q \neq q_0 \\ \Sigma^* X_q Y_q^* \cup Y_q^* & \text{if } q = q_0, \end{cases}$$

for all $q \in Q$. Lastly, let

$$E = \left(\bigcup_{q \in Q \setminus F} E_q \right) \cup \left(\bigcup_{q \in Q} E_q Z_q \Sigma^* \right).$$

Proof. We claim

$$u \in L(E_q) \implies q_0 u \subseteq \{q\} \tag{7}$$

and

$$q_0 u = \{q\} \implies u \in L(E_q), \tag{8}$$

for all $q \in Q$, $u \in \Sigma^*$. Then (6) follows since the definition of E expresses the fact that an input word $u \in \Sigma^*$ is rejected by the automaton $\mathcal{A}$ either if $q_0 u = \{q\}$ for some non-final state q, or $q_0 u = \emptyset$.

As (7) is quite obvious, we only prove (8). Suppose that $q_0 u = \{q\}$ for some state $q \in Q$ and input word $u \in \Sigma^n$, $n \geq 0$. Then there exist some states $q_1, \ldots, q_{n-1}$ such that

$$q_0 \xrightarrow{u_1} q_1 \xrightarrow{u_2} \cdots \xrightarrow{u_{n-1}} q_{n-1} \xrightarrow{u_n} q.$$

If $q_0 = q_1 = \cdots = q_{n-1} = q$ then $u \in Y_q^* \subseteq L(E_q)$. Otherwise let $k \in [n]$ be the largest index for which $q_{k-1} \neq q$, so that

$$q \neq q_{k-1} \xrightarrow{u_k} q \xrightarrow{u_{k+1}} \cdots \xrightarrow{u_{n-1}} q \xrightarrow{u_n} q.$$

Since $\mathcal{A}$ is deterministic it follows that $u_{k+1}, \ldots, u_n \in Y_q$. Moreover, since $q \neq q_{k-1} \xrightarrow{u_k} q$, the relation induced by u_k is not the identity function. Thus, u_k induces a partial constant function with range $\{q\}$, so that $u_k \in X_q$. We see $u \in \Sigma^* X_q Y_q^* \subseteq L(E_q)$. □

The next construction presents the main idea of reducing $\mathbf{AIP}_R$ to $\mathbf{ASF}_R$. The very same idea was used by Cho and Huynh in [3].

CONSTRUCTION 4.4 **Input:** A sequence $\mathcal{B}_1, \ldots, \mathcal{B}_n$ $(n \geq 2)$ of minimal reset automata of the form $\mathcal{B}_i = (Q_i, \Sigma, \tau_i, I_i, F_i)$.
Output: A minimal DFA $\mathcal{C}$ such that

$$\bigcap_{i \in [n]} L(\mathcal{B}_i) = \emptyset \iff L(\mathcal{C}) \text{ is star-free.} \tag{9}$$

Description: If $I_i = \emptyset$ for some $i \in [n]$ then let $\mathcal{C}$ be the minimal DFA with input alphabet $\{0\}$ recognizing the star-free language $\emptyset$. From now on we assume that each automaton $\mathcal{B}_i$ has a unique initial state s_i. Thus $L(B_i) \neq \emptyset$, for all $i \in [n]$. Let p be the least prime number with $p \geq n$. It is well known (see [6]) that $p < 2n$. For integers $i \in \{n+1, n+2, \ldots, p\}$ let $\mathcal{B}_i = (Q_i, \Sigma, \tau_i, \{s_i\}, F_i)$ be a minimal DFA recognizing the language Σ^*. For the sake of simplicity assume that the sets Q_i $(i \in [p])$ are pairwise disjoint, and that $\# \notin \Sigma$ is a new input symbol. Let $\nu : \mathbf{N} \to [p]$ be the function mapping each integer i to $((i-1) \bmod p) + 1$. Then we define

$$\mathcal{C} := (\bigcup_{i \in [p]} Q_i, \Sigma \cup \{\#\}, \tau, \{s_1\}, \{s_1\}),$$

where

$$\begin{aligned} \tau(\#) &= \bigcup_{i \in [p]} F_i \times \{s_{\nu(i+1)}\} \\ \tau(\sigma) &= \bigcup_{i \in [p]} \tau_i(\sigma), \end{aligned}$$

for all input symbols $\sigma \in \Sigma$. See Figure 1.

Proof. Clearly, $\mathcal{C}$ is a DFA with

$$L(\mathcal{C}) = \left(L(\mathcal{B}_1)\#L(\mathcal{B}_2)\# \cdots L(\mathcal{B}_n)\#(\Sigma^*\#)^{p-n}\right)^*.$$

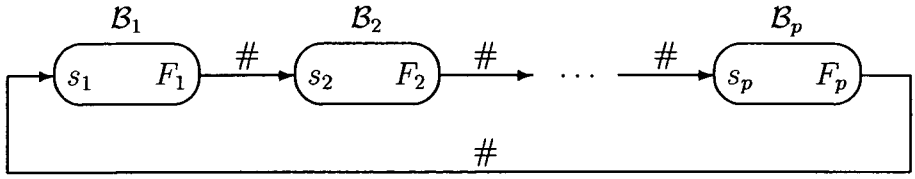


Figure 1: The automaton $\mathcal{C}$

By Schützenberger's theorem, (9) is equivalent to the condition

$$\bigcap_{i\in[n]} L(\mathcal{B}_i) \neq \emptyset \iff \mathcal{C} \text{ is not aperiodic.} \tag{10}$$

The "$\Longrightarrow$" part of (10) is obvious: if $u \in \Sigma^*$ is a common element of the languages $L(B_1), \ldots, L(\mathcal{B}_n)$ then $s_1 \xrightarrow{(u\#)^p}_{\mathcal{C}} s_1$ and $s_1 \xrightarrow{u\#}_{\mathcal{C}} s_2 \neq s_1$, so that $\mathcal{C}$ is not aperiodic by Remark 2.1. Before we prove the "$\Longleftarrow$" part of (10) observe that if the letter # appears l times in an input word $u \in (\Sigma \cup \{\#\})^*$, and $q \in Q_i$ is a state such that $q(u\#)_{\mathcal{C}} \neq \emptyset$ then $q(u\#)_{\mathcal{C}} = \{s_{\nu(i+l+1)}\}$. Moreover, if $s_j(v\#)_{\mathcal{C}} \neq \emptyset$ for some integer $j \in [p]$ and word $v \in \Sigma^*$ then $v \in L(\mathcal{B}_j)$. Now suppose that $\mathcal{C}$ is not aperiodic, i.e.

$$q \xrightarrow{u^k}_{\mathcal{C}} q, \tag{11}$$

and

$$q \xrightarrow{u}_{\mathcal{C}} q', \tag{12}$$

for some *different* states $q \in Q_i$, $q' \in Q_{i'}$, $i, i' \in [p]$, input word $u \in (\Sigma \cup \{\#\})^+$ and integer $k \geq 2$. Note that by (11) we have $q(u^t)_{\mathcal{C}} \neq \emptyset$, for all $t \geq 0$. Let l be the number of #'s in u, so that u can be written as

$$u^{(0)}\#u^{(1)}\#\ldots\#u^{(l)},$$

where $u^{(0)}, \ldots, u^{(l)}$ are words in Σ^*. If l were 0 then we would have $i' = i$, $q \xrightarrow{u^k}_{\mathcal{B}_i} q$, and $q \xrightarrow{u}_{\mathcal{B}_i} q' \neq q$, contradicting the fact that $\mathcal{B}_i$ is aperiodic. Thus, $l > 0$.

Let v denote the word $u^{(0)}\#\cdots\#u^{(l-1)}$, so that $u = v\#u^{(l)}$ and

$$q \xrightarrow{v\#}_{\mathcal{C}} s_j,$$

where $j = \nu(i + l)$. By (11) we have

$$q \xrightarrow{u^{k-1}v\#}_{\mathcal{C}} s_i \xrightarrow{u^{(l)}}_{\mathcal{C}} q.$$

If p were a divisor of l then it would follow that $j = i$ and $q \xrightarrow{v\#}_{\mathcal{C}} s_i \xrightarrow{u^{(l)}}_{\mathcal{C}} q$, contradicting (12). Thus p is not a divisor of l.

Let j be an arbitrary element of $[n]$. As p is a prime not dividing l, there exists some integer $t \geq 0$ such that $\nu(i + lt) = j$. For this t we have

$$q \xrightarrow{u^{t-1}v\#}_{\mathcal{C}} s_j.$$

Moreover, since $u^{t-1}v\#u^{(l)}u^{(0)}\#$ is a prefix of u^{t+1} and $q(u^{t+1})_{\mathcal{C}} \neq \emptyset$, it follows that

$$s_j(u^{(l)}u^{(0)}\#)_{\mathcal{C}} = q(u^{t-1}v\#u^{(l)}u^{(0)}\#)_{\mathcal{C}} \neq \emptyset,$$

showing $u^{(l)}u^{(0)} \in L(\mathcal{B}_j)$. Since $j \in [n]$ was arbitrary,

$$u^{(l)}u^{(0)} \in \bigcap_{j\in[n]} L(\mathcal{B}_j).$$

In order to prove $\mathcal{C}$ is minimal suppose that $q \in Q_j$ and $r \in Q_k$ are two different states of the automaton $\mathcal{C}$. For each $i \in [p]$ choose an arbitrary word $v^{(i)} \in L(\mathcal{B}_i)$. Since q is a biaccessible state of $\mathcal{B}_j$, there exist words $v, w \in \Sigma^*$ with

$$s_j \xrightarrow{v}_{\mathcal{B}_j} q \xrightarrow{w}_{\mathcal{B}_j} F_j.$$

Then

$$s_1 \xrightarrow{v^{(1)}\#\cdots\#v^{(j-1)}\#v}_{\mathcal{C}} q \xrightarrow{w\#v^{(j+1)}\#\cdots\#v^{(p)}\#}_{\mathcal{C}} s_1,$$

showing q is a biaccessible state of $\mathcal{C}$. If $j \neq k$, say $j < k$, then

$$\begin{aligned} q(w\#v^{(j+1)}\#\cdots\#v^{(p)}\#)_{\mathcal{C}} &= \{s_1\} \quad \text{and} \\ r(w\#v^{(j+1)}\#\cdots\#v^{(p)}\#)_{\mathcal{C}} &\subseteq \{s_{k-j+1}\}. \end{aligned}$$

Lastly, suppose that $j = k$. Since $\mathcal{B}_j$ is minimal, there exists some word $x \in \Sigma^*$ such that exactly one of the sets $qx_{\mathcal{B}_j} \cap F_j$ and $rx_{\mathcal{B}_j} \cap F_j$ is empty, say $rx_{\mathcal{B}_j} \cap F_j = \emptyset$. Then $q(x\#)_{\mathcal{C}} = \{s_{\nu(j+1)}\}$, $r(x\#)_{\mathcal{C}} = \emptyset$ and we have

$$\begin{aligned} q(x\#v^{(j+1)}\#\cdots\#v^{(p)}\#)_{\mathcal{C}} &= \{s_1\} \quad \text{and} \\ r(x\#v^{(j+1)}\#\cdots\#v^{(p)}\#)_{\mathcal{C}} &= \emptyset. \end{aligned}$$

□

The last construction gives the second part of the reduction $\mathbf{AIP}_R \leq_{log} \mathbf{ASF}_R$.

CONSTRUCTION 4.5 **Input:** A minimal DFA $\mathcal{C} = (Q, \Sigma, \tau, I, F)$.
Output: A minimal DFA $\mathcal{C}'$ with input alphabet $\{0, 1\}$ such that

$$L(\mathcal{C}) \text{ is star-free} \iff L(\mathcal{C}') \text{ is star-free.} \tag{13}$$

Description: Let $\psi : \Sigma^* \to \{0,1\}^*$ be an injective homomorphism satisfying the conditions of Lemma 2.1. In particular, the image $\psi(\sigma)$ of each symbol $\sigma \in \Sigma$ is a word in $\{0,1\}^{2l}$, where $l = \lceil \log_2(|\Sigma| + 1) \rceil$. For each state $q \in Q$ let

$$S_q \;:=\; \{\psi(\sigma)q' \mid \sigma \in \Sigma,\ q' \in Q,\ q \xrightarrow{\sigma}_C q'\},$$

so that S_q is a set of words over the alphabet $\{0,1\} \cup Q$, more precisely, $S_q \subseteq \{0,1\}^{2l}Q$. When S is a set of words and u is a word over the same alphabet, $u\backslash S$ denotes the set $\{v \mid uv \in S\}$. For each integer $j \in [2l-1]$ let

$$Q'_j \;:=\; \{u\backslash S_q \mid q \in Q,\ u \in \{0,1\}^j\} \setminus \{\emptyset\}.$$

Thus each element of Q'_j is a nonempty subset of $\{0,1\}^{2l-j}Q$. Now let

$$C' \;:=\; (Q \cup Q', \{0,1\}, \tau', I, F),$$

where

$$Q' \;=\; \bigcup_{j \in [2l-1]} Q'_j,$$

and τ' is defined such that

$$qx_{C'} \;=\; \begin{cases} \{x\backslash S_q\} & \text{if } x\backslash S_q \neq \emptyset, \\ \emptyset & \text{otherwise,} \end{cases}$$

$$Sx_{C'} \;=\; \begin{cases} \{q'\} & \text{if } x\backslash S = \{q'\}, \text{ for some } q' \in Q, \\ \emptyset & \text{if } x\backslash S = \emptyset, \\ \{x\backslash S\} & \text{otherwise,} \end{cases}$$

for all $q \in Q$, $S \in Q'$, $x \in \{0,1\}$.

Proof. Let us denote Q by Q'_0. It is easy to see that C' is a DFA satisfying

$$q \xrightarrow{u}_{C'} q' \iff \exists v \in \Sigma^*\ u = \psi(v) \wedge q \xrightarrow{v}_C q', \tag{14}$$

and

$$Q'_i \xrightarrow{u}_{C'} Q'_j \implies |u| \equiv j - i \pmod{2l}, \tag{15}$$

for all $q, q' \in Q$, $u \in \{0,1\}^*$, $0 \leq i, j < 2l$. It follows in particular that $L(C') = \psi(L(C))$, so that (13) holds by Lemma 2.1.

In order to prove C' is minimal suppose that $s \in Q'_i$ and $s' \in Q'_j$ $(0 \leq i, j < 2l)$ are two different states of C'. It is clear from the description of C' that there exist words $v \in \{0,1\}^i$, $v' \in \{0,1\}^{2l-i}$, and states $q, q' \in Q$ such that $q \xrightarrow{v}_{C'} s \xrightarrow{v'}_{C'} q'$. Since C is minimal, there exist words $u, u' \in \Sigma^*$ with $I \xrightarrow{u}_C q$ and $q' \xrightarrow{u'}_C F$. By (14) we have

$$I \xrightarrow{\psi(u)}_{C'} q \xrightarrow{v}_{C'} s \xrightarrow{v'}_{C'} q' \xrightarrow{\psi(u')}_{C'} F,$$

showing s is a biaccessible state of C'. If $i \neq j$ then s and s' are not equivalent by (15), so suppose $i = j$. If $i = j = 0$ then s and s' are two different elements of Q, and since C is minimal there exists a word $w \in \Sigma^*$ such that exactly one of the two sets $sw_C \cap F = s\psi(w)_{C'} \cap F$ and $s'w_C \cap F = s'\psi(w)_{C'} \cap F$ is empty. Lastly suppose that $i = j \in [2l-1]$. Then s and s' are two different subsets of the set $\{0,1\}^{2l-i}Q$, say $s \not\subseteq s'$. Let uq be an arbitrary element in s which is not in s', where $u \in \{0,1\}^{2l-i}$ and $q \in Q$. There are two possibilities: either $s'u_{C'} = \emptyset$ or $s'u_{C'} = \{q'\}$ for some state $q' \in Q$, $q' \neq q$. In the first case we have $su\psi(v)_{C'} \cap F \neq \emptyset$ and $s'u\psi(v)_{C'} \cap F = \emptyset$, where $v \in \Sigma^*$ is an arbitrary word with $q \xrightarrow{v}_C F$. (Such a v exists since q is a coaccessible state of C.) The second case can be handled similarly to the case $i = j = 0$. □

5 Results

THEOREM 5.1 *The problems* **AIP** *and* $\mathbf{AIP}_R$ *are* **PSPACE***-complete with respect to logspace reductions.*

Proof. We show

$$\mathbf{PSPACE} \leq_{log} \mathbf{AIP}_R \leq_{log} \mathbf{AIP} \in \mathbf{PSPACE}.$$

Suppose that $L \subseteq \Sigma^*$ is a language in **PSPACE**. Then there exists a polynomial function $p : \mathbf{N} \to \mathbf{N}$ and a deterministic Turing machine $\mathcal{M}$ of space-complexity p such that $L(\mathcal{M}) = L$. Applying Construction 4.1 followed by Construction 4.2 to $\mathcal{M}$ and an input word $u \in \Sigma^*$, we obtain a list $\mathcal{A}_1, \ldots, \mathcal{A}_n$ of minimal reset automata such that

$$u \in L \iff \bigcap_{i \in [n]} L(\mathcal{A}_i) \neq \emptyset.$$

Since both constructions can be carried out by a logspace-bounded Turing machine, $\mathbf{PSPACE} \leq_{log} \mathbf{AIP}_R$. The claim $\mathbf{AIP}_R \leq_{log} \mathbf{AIP}$ is trivial. In order to prove $\mathbf{AIP} \in \mathbf{PSPACE}$ suppose that $\mathcal{A}_1, \ldots, \mathcal{A}_n$ are NFA's with a common input alphabet Σ, say $\mathcal{A}_i = (Q_i, \Sigma, \tau_i, I_i, F_i)$. The following nondeterministic PASCAL-style program accepts the automata $\mathcal{A}_1, \ldots, \mathcal{A}_n$ if and only if $\bigcap_{i \in [n]} L(\mathcal{A}_i) \neq \emptyset$:

```
function Solve_AIP(A_1, ... A_n : NFA):boolean;
var
    S_1, ..., S_n : set of state;
    σ : input symbol;
begin
    S_1 := I_1;
      ⋮
    S_n := I_n;
```

```
    while S_1 ∩ F_1 = ∅ or ··· or S_n ∩ F_n = ∅ do
    begin
        guess σ ∈ Σ;
        S_1 := S_1 σ_{A_1};
            ⋮
        S_n := S_n σ_{A_n};
    end;
    Solve_AIP:=true;
end;
```

The space complexity of the program is linear. It follows by Savitch's theorem that **AIP** $\in$ **PSPACE**. □

THEOREM 5.2 *The problems* **ASF** *and* $\mathbf{ASF}_R$ *are* **PSPACE***-complete with respect to logspace reductions.*

Proof. We show

$$\overline{\mathbf{AIP}_R} \leq_{log} \mathbf{ASF}_R \leq_{log} \mathbf{ASF} \in \mathbf{PSPACE}.$$

Suppose that $\mathcal{B}_1, \ldots, \mathcal{B}_n$ $(n \geq 2)$ are minimal reset automata with a common input alphabet. Applying Construction 4.4 followed by Construction 4.5 to $\mathcal{B}_1, \ldots, \mathcal{B}_n$, we obtain a minimal DFA $\mathcal{C}'$ with input alphabet $\{0,1\}$ such that

$$\bigcap_{i \in [n]} L(\mathcal{B}_i) = \emptyset \iff L(\mathcal{C}') \text{ is star-free.}$$

Since both constructions can be carried out by a logspace-bounded Turing machine, $\overline{\mathbf{AIP}_R} \leq_{log} \mathbf{ASF}_R$. The claim $\mathbf{ASF}_R \leq_{log} \mathbf{ASF}$ is trivial. In order to prove **ASF** $\in$ **PSPACE** suppose that $\mathcal{A} = (Q, \Sigma, \tau, I, F)$ is an NFA. By Schützenberger's theorem, $L(\mathcal{A})$ is star-free if and only if the minimal DFA recognizing $L(\mathcal{A})$ is aperiodic. Recall that the power automaton of $\mathcal{A}$ is the deterministic automaton

$$\mathrm{P}(\mathcal{A}) = (\mathrm{P}(Q), \Sigma, \tau', \{I\}, F'),$$

where

$$\begin{aligned} F' &= \{S \in \mathrm{P}(Q) \mid S \cap F \neq \emptyset\}, \\ \tau'(\sigma) &= \{(S, S\sigma_{\mathcal{A}}) \mid S \in \mathrm{P}(Q)\}, \end{aligned}$$

for all $\sigma \in \Sigma$. The minimal DFA recognizing $L(\mathcal{A})$ is obtained from $\mathrm{P}(\mathcal{A})$ by deleting those states which are not biaccessible, and then identifying the equivalent states. It follows that $L(\mathcal{A})$ is star-free if and only if there exists some input word $u \in \Sigma^*$, *accessible* state S of $\mathrm{P}(\mathcal{A})$ and integer $k \geq 2$ such that $S \approx_{\mathrm{P}(\mathcal{A})} S(u^k)_{\mathcal{A}} = S(u_{\mathcal{A}})^k$ and $S \not\approx_{\mathrm{P}(\mathcal{A})} Su_{\mathcal{A}}$. The following nondeterministic procedure decides if $S \not\approx_{\mathrm{P}(\mathcal{A})} S'$ holds for two states S, S' of $\mathrm{P}(\mathcal{A})$:

```
function Not_Equiv(S, S' : set of state):boolean;
var
    σ : input symbol;
begin
    while (S ∩ F = ∅ and S' ∩ F = ∅) or
          (S ∩ F ≠ ∅ and S' ∩ F ≠ ∅) do
    begin
        guess σ ∈ Σ;
        S := Sσ_A;
        S' := S'σ_A;
    end;
    Not_Equiv:=true;
end;
```

By Savitch's theorem we obtain a deterministic polynomial-space program Equiv which decides if two states of $\mathrm{P}(\mathcal{A})$ are equivalent. The following nondeterministic program uses Equiv as a subroutine to decide if $L(\mathcal{A})$ is not star-free:

```
function Not_ASF(A : NFA):boolean;
var
    σ : input symbol;
    S, S' : set of state;
    ρ : relation;
    halt : boolean;
begin
    S := I;
    repeat
        guess σ ∈ Σ;
        S := Sσ_A;
        guess halt;
    until halt;
    ρ := ε_A;
    repeat
        guess σ ∈ Σ;
        ρ := ρ ∘ σ_A;
        guess halt;
    until halt;
    S' := Sρ;
    if Equiv(S, S') then
        Not_ASF:=false
    else begin
        repeat
            S' := S'ρ;
        until Equiv(S, S');
        Not_ASF:=true;
    end;
```

`end;`

By Savitch's theorem and the fact that the language class **PSPACE** is closed under complementation it follows that $\mathbf{ASF} \in \mathbf{PSPACE}$. □

THEOREM 5.3 *The problems* **RSF** *and* $\mathbf{RSF}_R$ *are* **PSPACE***-complete with respect to logspace reductions.*

Proof. We show

$$\overline{\mathbf{AIP}_R} \leq_{log} \mathbf{RSF}_R \leq_{log} \mathbf{RSF} \leq_{log} \mathbf{ASF}.$$

The claim $\mathbf{RSF}_R \leq_{log} \mathbf{RSF}$ is trivial, and it is also easy to see that $\mathbf{RSF} \leq_{log} \mathbf{ASF}$: given a regular expression E, a logspace-bounded Turing machine can construct a *nondeterministic* automaton $\mathcal{A}$ such that $L(E) = L(\mathcal{A})$.

Suppose that $\mathcal{B}_1, \ldots, \mathcal{B}_n$ $(n \geq 2)$ are minimal reset automata with a common input alphabet Σ. Let $\mathcal{C}$ be the result of Construction 4.4 applied to the automata $\mathcal{B}_1, \ldots, \mathcal{B}_n$. Then $\mathcal{C}$ is a minimal DFA with input alphabet $\Sigma \cup \{\#\}$ such that

$$\bigcap_{i \in [n]} L(\mathcal{B}_i) = \emptyset \iff L(\mathcal{C}) \text{ is star-free.}$$

Applying Construction 4.3 to each one of the automata $\mathcal{B}_1, \ldots, \mathcal{B}_n$ we get regular expressions $E_1, \ldots, E_n$ such that

$$L(E_i) = \overline{L(\mathcal{B}_i)},$$

for all $i \in [n]$. Recall that

$$L(\mathcal{C}) = \left(L(\mathcal{B}_1)\#L(\mathcal{B}_2)\# \cdots L(\mathcal{B}_n)\#(\Sigma^*\#)^{p-n}\right)^*,$$

where p is the least prime number with $p \geq n$. It follows that a word $v = v^{(0)}\#v^{(1)}\# \cdots v^{(k-1)}\#v^{(k)}$ $(k \geq 0,\ v^{(0)}, \ldots, v^{(k)} \in \Sigma^*)$ belongs to $L(\mathcal{C})$ if and only if $v^{(k)} = \epsilon$, k is a multiple of p, and $v^{(i)} \in L(\mathcal{B}_{(i \bmod p)+1})$, for all $i < k$ with $i \bmod p < n$. The languages denoted by the regular expressions

$$\begin{aligned} F_1 &= (\Sigma \cup \#)^*\Sigma \\ F_2 &= ((\Sigma^*\#)^p)^* \left(\bigcup_{i \in [p-1]} (\Sigma^*\#)^i \right) \Sigma^* \\ F_3 &= ((\Sigma^*\#)^p)^* \left(\bigcup_{i \in [n]} (\Sigma^*\#)^{i-1} E_i \# \right) (\Sigma \cup \#)^* \end{aligned}$$

consist of those words $v = v^{(0)}\#v^{(1)}\# \cdots v^{(k-1)}\#v^{(k)}$ for which

1. $v^{(k)} \neq \epsilon$,
2. k is not a multiple of p,
3. $v^{(i)} \notin L(B_{(i \bmod p)+1})$ for some $i < k$ with $i \bmod p < n$,

respectively. Thus, the regular expression $E := F_1 \cup F_2 \cup F_3$ denotes the complement of the language $L(\mathcal{C})$. Let $\psi : (\Sigma \cup \{\#\})^* \to \{0,1\}^*$ be a homomorphism satisfying the conditions of Lemma 2.1. Let E' be the regular expression obtained from E by replacing each occurence of every letter $x \in \Sigma \cup \{\#\}$ by the word $\psi(x) \in \{0,1\}^*$. Then E' is a regular expression over the alphabet $\{0,1\}$ having star-height 2. Moreover, $L(E') = \psi(L(E)) = \psi(\overline{L(\mathcal{C})})$, so that

$$L(E') \text{ is star-free} \iff L(\mathcal{C}) \text{ is star-free} \iff \bigcap_{i \in [n]} L(\mathcal{B}_i) = \emptyset.$$

The simple structure of E' assures that it can be constructed by a logspace-bounded Turing machine. □

6 Open problems

The above results suggest that the following questions may be interesting.

1. What is the complexity of deciding whether $\bigcap_{i \in [n]} L(\mathcal{A}_i) \neq \emptyset$, for minimal complete reset automata $\mathcal{A}_1, \ldots, \mathcal{A}_n$?
2. What is the complexity of deciding whether a regular expression of star-height 1 denotes a star-free language?

We conjecture that the answer for the first question is "**NP**-complete".

The second question seems to be harder. However, it is our conjecture that restricting the problem **RSF** to regular expressions of star-height 1 substantially decreases its computational complexity.

7 Acknowledgement

I would like to thank Zoltán Ésik for encouragement and for many useful comments and suggestions. I also thank Stephen L. Bloom and an anonymous referee whose suggestions have been incorporated.

References

[1] L. Bernátsky and Z. Ésik. Semantics of Flowchart Programs and the Free Conway Theories. Submitted for publication to RAIRO Theoretical Informatics and Applications.

[2] D. P. Bovet and P. Crescenzi. *Introduction to the Theory of Complexity.* Prentice Hall, 1994.

[3] Sang Cho and Dung T. Huynh. Finite-automaton aperiodicity is PSPACE-complete. *Theoretical Computer Science*, 88:99–116, 1991.

[4] Samuel Eilenberg. *Automata, Languages and Machines.* Academic Press, New York and London, 1974.

[5] Michael R. Garey and David S. Johnson. *Computers and intractability, A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, New York, 1979.

[6] G.H. Hardy and E.M. Wright. *An Introduction to the Theory of Numbers.* Oxford University Press, London, 3rd edition, 1954.

[7] A. Salomaa. *Theory of Automata.* Pergamon Press, 1969.

[8] M. P. Schützenberger. On Finite Monoids Having Only Trivial Subgroups. *Information and Control*, 8:190–194, 1965.

[9] Jacques Stern. Complexity of Some Problems from the Theory of Automata. *Information and Control*, 66:163–176, 1985.

[10] H. Straubing. *Finite Automata, Formal Languages and Circuit Complexity.* Birkhäuser, 1994.

A Appendix

Proof of Lemma 2.1. First of all note that $n \leq 2^l - 1$, so that l bits are sufficient to represent the number n in binary. Let $\psi : \Sigma^* \to \{0,1\}^*$ be the homomorphism mapping each letter $\sigma_i \in \Sigma$ ($i \in [n]$) to the $2l$-bit binary representation of i. Then ψ is injective and satisfies (1).

Proof of (2). Suppose that (2) is not true. Then there exist some words $u, v, w \in \{0,1\}^*$ such that $uv^{2l}w \in \psi(\Sigma^*)$, but $2l$ is not a divisor of $|v|$. Let us denote $|v|$ by m. Then $m > 0$ and $\gcd(2l, m) < 2l$. Since none of the integers $l+1, l+2, \ldots, 2l-1$ is a divisor of $2l$,

$$\gcd(2l, m) \quad \leq \quad l. \tag{16}$$

Moreover, since no word in $\psi(\Sigma^*)$ may contain 0^{2l} as a subword, the letter 1 occurs in the word v^{2l}, and thus in v. Let $j \in [m]$ be an integer such that the jth letter of v is 1. If $i \in [2lm]$ is an integer satisfying

$$i \quad \equiv \quad j \pmod{m} \tag{17}$$

then the ith letter of v^{2l} is 1. By (1) it follows that if $1 \leq i \leq |uv^{2l}w|$ is an integer such that $(i-1) \bmod 2l < l$, then the ith letter of $uv^{2l}w$ is 0. In other words, if $i \in [2lm]$ is an integer satisfying

$$i \equiv t - |u| \pmod{2l} \tag{18}$$

for some $t \in [l]$, then the ith letter of v^{2l} is 0. The diophantic system (17,18) is solvable in the variable i if and only if

$$t - |u| \equiv j \pmod{\gcd(2l, m)}, \tag{19}$$

and in this case every solution can be written in the form

$$i = i_0 + h \cdot \operatorname{lcm}(2l, m),$$

where i_0 is a fixed solution and h is an integer. Let t be the unique element of $[\gcd(2l, m)]$ satisfying (19). Then $t \in [l]$, by (16). For this t there exists a unique integer $i \in [\operatorname{lcm}(2l, m)] \subseteq [2lm]$ such that both (17) and (18) hold. But then we have the contradiction that the ith letter of v^{2l} is equal to both 0 and 1. This contradiction was caused by the assumption that (2) fails.

Proof of (3). Suppose that $L \subseteq \Sigma^*$ is a language and $\psi(L) \subseteq \{0,1\}^*$ is star-free. Then L is regular and there exists an integer $k \geq 0$ such that for all words $u, v, w \in \Sigma^*$,

$$\begin{aligned} uv^k w \in L &\iff \psi(u)\psi(v)^k\psi(w) \in \psi(L) \\ &\iff \psi(u)\psi(v)^{k+1}\psi(w) \in \psi(L) \\ &\iff uv^{k+1}w \in L, \end{aligned}$$

showing L is star-free. Thus, for this direction no special property of the homomorphism ψ is needed other than its injectivity.

For the converse direction, suppose that $L \subseteq \Sigma^*$ is a star-free language. Then there exists an integer $k \geq 0$ such that

$$xy^k z \in L \iff xy^{k+1}z \in L, \tag{20}$$

for all words $x, y, z \in \Sigma^*$. Let m be the maximum of $2l$ and $k+1$. Suppose that $uv^m w \in \psi(L)$, for some $u, v, w \in \{0,1\}^*$. We want to show that $uv^{m+1}w \in \psi(L)$. This is obvius if $|v| = 0$, so suppose that $|v| > 0$. By (2) it follows that $|v|$ is a multiple of $2l$, so that $|v| \geq 2l$. Let α be the shortest prefix of v such that the length of the word $u\alpha$ is a multiple of $2l$. Then v can be written as $\alpha\beta$, for some word $\beta \in \{0,1\}^*$. Since $uv^m w = u\alpha(\beta\alpha)^{m-1}\beta w \in \psi(L)$ and the length of the words $u\alpha$, $\beta\alpha$ and βw are multiples of $2l$, there exist words $x, y, z \in \Sigma^*$ such that $\psi(x) = u\alpha$, $\psi(y) = \beta\alpha$, $\psi(z) = \beta w$ and $xy^{m-1}z \in L$. Since $m - 1 \geq k$, it follows by (20) that $xy^m z \in L$. Thus,

$$\psi(xy^m z) = u\alpha(\beta\alpha)^m\beta w = uv^{m+1}w \in \psi(L).$$

The implication $uv^{m+1}w \in \psi(L) \Longrightarrow uv^m w \in \psi(L)$ is proved in a similar way. □

Received January, 1996

Acta Cybernetica **13** (1997) 23-39.

Server Problems and Regular Languages

B. Csaba * G. Dányi †

Abstract

The sequences of requests are considered as words over the alphabet of vertices. We assume that the server problem is restricted, meaning that the request words are chosen from a subset of all possible words, i.e. from a language. We define the class $ONLINE$ consisting of the languages, for which there exists a 1-competitive satisfying on-line algorithm. Our main result is a sufficient condition for languages to be in $ONLINE$ and a construction method of 1-competitive on-line algorithm for the ones, which satisfy that condition. We perform this by characterizing a subclass $ONREG_0$ of the class $ONLINE \cap REG$, where REG is the class of regular languages. Moreover, we prove some results, which help to show the on-lineness of certain other (even nonregular) languages and we give sufficient conditions to prove that a language is not on-line.

1 Introduction

The k-server problem is a generalized model of certain scheduling problems as, for instance, multi-level memory paging, disk caching and head motion planning of multi-headed disks (see [MMS]). The paging and caching problems have been studied for a long time. However, server problems are introduced in the 80's (see [ST] and [MMS]).

The k-server problem can be stated as follows. Let $M = (V, \delta)$ be a finite metric space, where $V = \{v_1, \ldots, v_n\}$ are the vertices and δ is the distance function. There are k mobile servers occupy exactly k vertices of M. Repeatedly a request, $v_i \in V$ appears. The request should be satisfied by moving some servers resulting that a server appears on the point v_i. The cost of moving one server from v_i to v_j is $\delta(v_i, v_j)$ and the cost of a satisfaction is the sum of the costs of the taken movements.

*Research of this author was supported by the Research Foundation of Hungary under Grant F4204.

Dept. of Computer Science, József Attila University, H-6701 Szeged, P.O.Box 652, Hungary, email: csaba@inf.u-szeged.hu

†Research of the author was supported by the Research Foundation of Hungary under Grant F012852 and by the Hungarian Cultural and Educational Ministry under Grant 434/94.

Dept. of Foundations of Computer Science, József Attila University, H-6701 Szeged, P.O.Box 652, Hungary, email: danyi@inf.u-szeged.hu

We assume that the request sequences are finite. The goal is to find an algorithm for M, which can satisfy request sequences with as little cost as possible.

If an algorithm serves requests immediately without knowing what the future requests will be, then we say that it is on-line. A widely used measure for the performance of an on-line algorithm is the competitive ratio, introduced by [ST]. Denote the optimal cost of the satisfaction of a request sequence x by $\mathrm{opt}(x)$. An on-line algorithm $\mathcal{A}$ is called c-competitive, if there exists a number K such that, for all allowed request sequences x, the total cost $\mathcal{A}(x)$, incurred by $\mathcal{A}$ on x, is at most $c\,\mathrm{opt}(x) + K$.

Obviously, a finite request sequence can be considered as a word over the alphabet V. It has been proved that, if the request sequences can be arbitrarily chosen, that is, any word in V^*, then the competitive ratio of any on-line algorithm is at least k (see [MMS]). However, in practice the request sequences are generated by programs, hence these sequences usually cannot be arbitrary, i.e. they are chosen from a language $L \subseteq V^*$. Then the server problem is said to be restricted. Knowing that language, we may expect to find on-line algorithms with better performance. We can assign competitive ratio to the languages, too. Actually, a language is called c-competitive, if, for any distance function δ, there exists c-competitive on-line algorithm satisfying its request sequences. Observe that the competitive ratio defines a hierarchy of language classes.

In this paper we consider the class $ONLINE$ consisting of 1-competitive languages, called on-line languages. Note that this class is the bottom element of the hierarchy mentioned above. However, $ONLINE$ seems to be very hard to characterize. For instance, it contains languages, which are event not recursively enumerable. For that very reason, we looked for necessary and sufficient conditions a language being in $ONLINE$.

Sufficient conditions can be found, if we consider appropriate subclasses of $ONLINE$. Such subclass can be defined, for example, by intersecting $ONLINE$ with a well known language class. In this paper we choose the class REG of regular languages for this purpose. This class is easy to handle, since the regular languages are recognized by deterministic finite automata. On the other hand, REG is closed for the operations concatenation, union and closure, which corresponds naturally to the programming structures, namely the sequencing, the selection, and the iteration, respectively. Observe that, for any alphabet V, $V^* \in REG$ holds, hence $REG \not\subseteq ONLINE$ follows. We define the subclass $ONREG_0 \subseteq REG$. The class $ONREG_0$ is closed for concatenation, union and closure of singleton languages. Roughly speaking, if we consider programming structures, there cannot be a selection inside an iteration. Our main result is that $ONREG_0 \subseteq ONLINE$. Moreover, we show that how the operations sublanguage construction and letter reduction can help to prove the on-lineness of certain other, even nonregular languages, and we give sufficient conditions to prove that a language is not on-line.

The outline of our paper is as follows. In the second section we introduce the definitions and notations, which are necessary to understand the paper. Moreover, we give some basic results. Our main result can be found in the third section, in which we show that the language class $ONREG_0$ contains on-line languages. In

the fourth section we give some sufficient conditions for regular languages not being on-line, and we discuss some other properties of the class $ONLINE$.

We note that there is an other way of studying restricted k-server problems, where the movements of the servers are restricted (see [FK], [BIRS]). This leads to the use of an access graph. A server can be moved to a vertex from an other one immediately, if they are adjacent vertices of the access graph. However, it can be seen that an access graph also defines a language over V, namely the set of satisfiable request sequences.

Acknowledgement. The authors are grateful to P. Hajnal (Dept. of Mathematics, University of Szeged, Hungary) and László Bernátsky (Dept. of Computer Science, University of Szeged, Hungary) for their valuable comments and suggestions.

2 Preliminaries

In this section we introduce the notions and notations which are necessary to understand the paper. Moreover, we recall the preliminaries referred in our proofs from other papers, and give some basic results.

We denote the set of real numbers by $\mathbf{R}$, the set of nonnegative real numbers by $\mathbf{R}^+$ and the set of natural numbers by ω. If H is a set, then $|H|$ denotes its cardinality.

We frequently use the principline of structural induction in our proofs. For more information about inductions see, for example, [W].

2.1 Languages and automata

Words and languages. An *alphabet* V is a finite nonempty set of symbols. The elements of an alphabet are called *letters* and denoted by u and v in this paper.

A *word* w over an alphabet V is a finite sequence $v_1 \dots v_l$ of some letters in V. The *length* of a word w, denoted by $|w|$, is the number of the letters composing w. The *empty string* is denoted by e, thus $|e| = 0$. For $w \in V^* - \{e\}$, we define $\mathrm{first}(w) \in V$ and $\mathrm{last}(w) \in V$ as the first and the last letter of w, respectively.

The *concatenation* w_1w_2 of two words $w_1 = v_1 \dots v_l$ and $w_2 = u_1 \dots u_k$ is the sequence $v_1 \dots v_l u_1 \dots u_k$. We define the powers of a word w as $w^0 = e$ and $w^n = w^{n-1}w$, for any integer $n \geq 1$. We say that a word w_1 is a *prefix* of a word w, and denote this fact by $w_1 \sqsubseteq w$, if there exists a word w_2, called a *suffix* of w, such that $w = w_1w_2$. We use the symbols w, x and y to denote words in this paper.

The set of all finite words over an alphabet V is denoted by V^*. A *language* over V is a subset $L \subseteq V^*$. For any languages L and L', we define their concatenation as $LL' = \{ww' \mid w \in L, w' \in L'\}$. For any language L, we put $L^0 = \{e\}$ and $L^i = L^{i-1}L$, where $i \geq 1$. The *closure* of a language L is the set $L^* = \cup_{i \in \omega} L^i$. The *prefix language* of a language L is $L^{\sqsubseteq} = \{x \mid x \sqsubseteq w \text{ holds for some } w \in L\}$. We say that the prefix problem is decidable for a language L over an alphabet V, if, for an arbitrary word $w \in V^*$, it is decidable whether $w \in L^{\sqsubseteq}$.

We define the operation *letter reduction*, denoted by lr, over words as follows. For any word $w \in V^*$, there exists a unique decomposition $w = v_1^{n_1} \dots v_k^{n_k}$, where $n_1, \dots, n_k \geq 1$ and $v_1, \dots, v_k \in V$, such that $v_i \neq v_{i+1}$ with $1 \leq i < k$. Then $\mathrm{lr}(w) = v_1 \dots v_k$. Roughly speaking, the lr operation substitutes a sequence of a letter by one. We extend the lr operation for languages as $\mathrm{lr}(L) = \cup_{w \in L} \mathrm{lr}(w)$.

Regular languages. The class *REG* of *regular languages* is the smallest class, which obeys the following rules.

(1) For any alphabet V, if $w \in V^*$ then $\{w\}$ is in *REG*.

(2) If $L, L' \in REG$ then $L \cup L'$, LL' and L^* are in *REG*.

That is, *REG* is the smallest class, which contains every singleton language and closed for the closure, the finite union and the concatenation. Clearly, for any alphabet V, the language V^* is regular.

The characterization of *REG* can be found in a wide range of books and papers concerning automata theory and formal languages (e.g. [HU]). A convenient way to define a regular language is to give its construction according to the above construction rules (1) – (2). This yields an expression, possibly with parentheses, where the members are singleton languages, and the operators are the union, the concatenation and the closure, in growing precedence order. These expressions are called *regular expressions*. For example, the regular expression $(\{a\} \cup \{b\})^*c$ defines the language of sequences of a and b followed by c. For brevity, we often write simply w for a singleton language $\{w\}$ in regular expressions in the sequel. Thus we have $(a \cup b)^*c$ for the above expression. Note that a regular language can be defined by several regular expressions.

Finite automata. A *deterministic finite automaton* (DFA) is a 5-tuple $A = (Q, V, \tau, q_0, F)$, where Q is the finite nonempty set of *states*, V is the *input alphabet*, $\tau : Q \times V \to Q$ is the total *transition function*, $q_0 \in Q$ is the *initial state* and $F \subseteq Q$ is the set of *final states*. We extend τ for $Q \times V^*$ with $\tau(q, e) = q$ and $\tau(q, wv) = \tau(\tau(q, w), v)$, where $q \in Q$, $w \in V^*$ and $v \in V$. A state $q \in Q$ is called *trap state* if there exists no word $w \in V^*$ such that $\tau(q, w) \in F$. We assume every state to be *accessible*, that is, for each $q \in Q$, there exists a word $w \in V^*$ such that $\tau(q_0, w) = q$.

A DFA can be represented as a directed labeled graph, where the vertices are the states and the edges are the transitions labeled by the corresponding letters of V. It should be clear that, for any $q, q' \in Q$ and $w \in V^*$, $\tau(q, w) = q'$ implies that there is a path from q to q' labeled by w. By the graph representation, it is easy to show that, for any state $q \in Q$, it can be decided in $O(|Q||V|)$ time, whether q is a trap state. Hence, the subset $T \subseteq Q$ of trap states can be determined in $O(|Q|^2|V|)$ time.

We say that the DFA $A = (Q, V, \tau, q_0, F)$ *recognizes* the word $w \in V^*$ if $\tau(q_0, w) \in F$. The recognizability of any word $w \in V^*$ by A can be decided in $O(|w|)$ time. The language recognized by A is the set $L_A = \{w \in V^* \mid \tau(q_0, w) \in F\}$. We say that the language L is *recognizable* if there exists a DFA A such that $L = L_A$. The following proposition, known as *Kleene's theorem*, is a fundamental result in the theory of automata.

Propositon 2.1 *A language is regular if and only if it is recognizable.*

Moreover, given a regular expression E, a DFA recognizing the language defined by E can be constructed effectively. Conversely, given a DFA A, a regular expression defining L_A can be constructed effectively, too (see [HU]).

Prefixes. The prefix words and languages play very important role in this paper, hence we pay more attention to them. Let $A = (Q, V, \tau, q_0, F)$ be a DFA. Observe that, for any word $w \in V^*$, $w \in L_A^{\sqsubseteq}$ holds if and only if $\tau(q_0, w)$ is not a trap state. Recall that the set of trap states $T \subseteq Q$ can be determined in $O(|Q|^2|V|)$ time. Define the DFA $B = (Q, V, \tau, q_0, Q - T)$, then it should be clear that $L_B = L_A^{\sqsubseteq}$. By Proposition 2.1, we have that if L is a regular language, then $L^{\sqsubseteq}$ is also regular. Hence, the prefix problem is decidable for regular languages.

2.2 The language class $ONREG_0$

We define a subclass $ONREG_0$ of REG. The name $ONREG_0$ refers to certain properties concerning server problems, which are explained later. The class $ONREG_0$ is investigated for the first time in the present paper, hence, in addition to the definition, it is necessary to characterize it. We do this by presenting three different definitions for $ONREG_0$ and proving their equivalence. The first definition shows the inclusion $ONREG_0 \subset REG$ immediately.

Definition 2.2 *The class $ONREG_0$ is the smallest one, which obeys the following rules.*

(1) *For any alphabet V, if $w \in V^*$ then $\{w\}$ and $\{w\}^*$ are in $ONREG_0$.*
(2) *If $L, L' \in ONREG_0$ then LL' and $L \cup L'$ are in $ONREG_0$.*

That is, $ONREG_0$ is the smallest class, which contains every singleton language, the closure of each singleton language, and closed for the finite union and the concatenation. Roughly speaking, a language $L \in ONREG_0$ can be constructed on the same way as a regular one, with the constraint that the closure operation is allowed only for singleton languages. It can be shown that, if V is an alphabet and $|V| > 1$, then $V^* \notin ONREG_0$. The second definition is very useful to prove the main result of the paper.

Definition 2.3 *$ONREG_0$ is the smallest class, which satisfies the following conditions.*

(1) *For any alphabet V and $w \in V^*$, $\{w\}, \{w\}^* \in ONREG_0$.*
(2) *For any alphabet V, $L \in ONREG_0$ and $w \in V^*$, $L\{w\}, L\{w\}^* \in ONREG_0$.*
(3) *If $L, L' \in ONREG_0$ then $L \cup L' \in ONREG_0$.*

The third definition describes explicitly, what kind of languages belongs to $ONREG_0$.

Definition 2.4 *Let V be an arbitrary alphabet and let L be a language over V. Then $L \in ONREG_0$ holds if and only if L can be defined by a regular expressions of the following form. There exist integers $r, s \geq 0$ and words $x_{i,j}, y_{i,j} \in V^*$, where $1 \leq i \leq r$ and $1 \leq j \leq s$, such that*

$$L = \cup_{1 \leq i \leq r} x_{i,1}(y_{i,1})^* \dots x_{i,j}(y_{i,j})^* \dots x_{i,s}(y_{i,s})^*.$$

Finally, we prove that the three definitions are equivalent. We perform this showing that Definition 2.4 is equivalent to both the definitions 2.2 and 2.3. Suppose that a language L can be defined by a regular expression of the form as in Definition 2.4. Then it is easy to see that L can be constructed by the rules of either Definition 2.2 or Definition 2.3. On the other hand, it is a routine exercise to show by structural induction on the rules that any language obeying the rules of either Definition 2.2 or Definition 2.3 can be defined by a regular expression of the form as in Definition 2.4.

2.3 Server problems and satisfying algorithms

Let $M = (V, \delta)$ be a finite metric space, where $V = \{v_1, \dots, v_n\}$ is the set of points and $\delta : V \times V \to \mathbf{R}^+$ is the distance function. Thus, for any $u, v, v' \in V$, $\delta(u, v) = \delta(v, u)$ and $\delta(u, v) \leq \delta(u, v') + \delta(v', v)$ hold. Moreover, $\delta(u, v) = 0$ if and only if u and v are identical. Note that M can be represented as an n-vertex complete graph G_M, where the vertices are labeled by the elements of V and the edges are weighted as determined by δ. We put $\delta_{max} = \max_{1 \leq i,j \leq n} \delta(v_i, v_j)$.

There are k mobile *servers* occupy exactly k vertices of G_M. We assume that $1 < k < n$ and $n \geq 3$. Note that the other cases are trivial and irrelevant from the point of view of our paper (see later). Suppose that there is a server on the vertex v_i and there is no one on v_j. Then moving the server from v_i to v_j costs $\delta(v_i, v_j)$. Let the metric space and the number of servers be arbitrary, but fixed in the sequel. We use the symbols n and k to denote the number of vertices and the number of servers, respectively.

A *configuration* (or a *state*) σ is a word $i_1 \dots i_k \in \{1, \dots, n\}^k$, where $i_j < i_{j+1}$ holds for each $1 \leq j < k$, showing that the servers are on the vertices $v_{i_1}, \dots, v_{i_k}$. It is easy to see that there are exactly $\binom{n}{k}$ different configurations. A configuration can be changed by moving a server from an occupied vertex to an empty one.

A *request* is a letter $v_i \in V$. A *satisfaction* of the request in a given *starting configuration* is a sequence of movements of some servers, such that the resulting configuration contains i, that is, there is a server on the vertex v_i. The cost of the satisfaction is the sum of the costs of its movements. Observe that a request v_i can be satisfied with no movements if and only if the starting configuration contains i. Moreover, any request can be satisfied by one movement of one server in any starting configuration.

A *request word* is a sequence of requests, that is a word over V. A satisfaction of a request word $w = v_1 \dots v_l$ in a given starting configuration is the sequence of the satisfactions of the requests $v_1, \dots, v_l$ after each other. The cost of a satisfaction is the sum of the costs of the satisfactions of composing requests.

Satisfactions are denoted by the symbol S in the sequel. By $|S|$ we mean the cost of the satisfaction S. For any satisfaction S, we denote the starting and the resulting configuration of S by σ_S and $\overline{\sigma}_S$, respectively. A *decomposition of the satisfaction* S of a word w is a sequence $S_1, \ldots, S_l$ of satisfactions of words $w_1, \ldots, w_l$, such that $w = w_1 \ldots w_l$, $\sigma_{S_1} = \sigma_S$ and $\overline{\sigma}_{S_i} = \sigma_{S_{i+1}}$ $(1 \leq i < l)$ hold, and the satisfaction of w by applying $S_1, \ldots, S_l$ after each other gives exactly S. Then we write $S = S_1 \ldots S_l$.

Clearly, for any request word w, there is a satisfaction with minimal cost in a given starting configuration σ, called an *optimal satisfaction* of w. Denote that cost by $\mathrm{opt}_\sigma(w)$. Let σ and σ' be different configuration, then it should be obvious that $|\mathrm{opt}_\sigma(w) - \mathrm{opt}_{\sigma'}(w)| \leq k\delta_{max}$. If the starting configuration σ is understood, then we write simply $\mathrm{opt}(w)$. Let the starting configuration be arbitrary, but fixed in the sequel.

Let $w \in V^*$ be a word such that $w = w_1 v^n w_2$ holds, for some $w_1, w_2 \in V^*$, $v \in V$ and $n \geq 1$. Then it is easy to show that $\mathrm{opt}(w_1 v^n w_2) = \mathrm{opt}(w_1 v w_2)$. It follows that, for any word w, $\mathrm{opt}(w) = \mathrm{opt}(\mathrm{lr}(w))$ holds.

The following results characterize the word composition and decomposition from the point of view of optimal satisfactions.

Lemma 2.5 *Let $w \in V^*$. Consider an arbitrary decomposition $w = w_1 w_2$, where $w_1, w_2 \in V^*$. Then the following statements hold.*

(1) $opt(w_1) + opt(w_2) \leq opt(w) + k\delta_{max}$

(2) $opt(w) \leq opt(w_1) + opt(w_2) + k\delta_{max}$

Proof. There exist satisfactions, denoted by S, S_1 and S_2, with the costs $\mathrm{opt}(w)$, $\mathrm{opt}(w_1)$ and $\mathrm{opt}(w_2)$ for the request words w, w_1 and w_2, respectively, in the starting configuration σ.

The satisfaction S can be decomposed as $S = S'S''$, where S' is a satisfaction of w_1 in σ, and S'' is a satisfaction of w_2 in $\overline{\sigma}_{S'}$. Observe that it is easy to transform S'' to a satisfaction of w_2 in σ such that, before starting S'', we change the starting configuration σ to $\overline{\sigma}_{S'}$ by moving the appropriate servers. Clearly, this modification costs at most $k\delta_{max}$. Now suppose the contrary of (1), that is, $\mathrm{opt}(w_1) + \mathrm{opt}(w_2) > \mathrm{opt}(w) + k\delta_{max}$. Then, by the above results, it is easy to see that either S' must be a more optimal satisfaction of w_1 than S_1 in σ, or S'' with the above modification should cost less than S_2 in σ. Both contradict that S_1 and S_2 have optimal costs, hence the statement (1) holds.

Now consider the following satisfaction of w in σ. Satisfy the prefix w_1 by S_1, then recover the starting configuration σ and satisfy the suffix w_2 by S_2. The satisfaction S_1 costs $\mathrm{opt}(w_1)$, the recovery of the starting configuration costs at most $k\delta_{max}$ and S_2 costs $\mathrm{opt}(w_2)$. Hence, the cost of the satisfaction is at most $\mathrm{opt}(w_1) + \mathrm{opt}(w_2) + k\delta_{max}$, which implies (2). □

Roughly speaking, for a given metric space and number of servers, our goal is to find satisfactions with minimal costs for request words. More precisely, we want to find an algorithm, which gives server moving sequences with minimal cost to satisfy any possible request word.

In this paper we consider such kind of algorithms, which are deterministic and computes the satisfaction of any request word in any starting configuration, that is, the server moving sequence of the satisfactions letter by letter. For the sort, we call simply *algorithm* the ones in the sequel, which have the above properties.

Let $\mathcal{A}$ be an algorithm, let w be a request word and let σ be the starting configuration. Then the cost of the satisfaction of w given by $\mathcal{A}$ in σ is denoted by $\mathcal{A}_\sigma(w)$. If σ is understood, we write simply $\mathcal{A}(w)$.

We say that an algorithm is *lazy*, if, for any request and starting configuration, it moves at most one server to satisfy the request. Otherwise, the algorithm is said *eager*. Clearly, for any request word, there exists a lazy algorithm which satisfies it. The following statement has been shown in [MMS].

Propositon 2.6 *For every eager algorithm, there exists a lazy one such that, for any request word, the satisfaction given by the lazy one costs no more than the satisfaction given by the eager one.*

This result provides that it is enough to find an eager algorithm, if we want to show the existence of a lazy one with respect to any cost limit. If the type of an algorithm is not defined explicitly, we mean eager one in the sequel.

There is an another classification of algorithms. An algorithm is said *off-line*, if it reads the whole request word first, then computes a satisfaction of that one. Moreover, an algorithm is called *on-line*, if it reads the request words from left to right, and reading a letter it gives the satisfaction of that request before reading the next one.

Let $c \geq 1$ be a real number and let $L \subseteq V^*$ be a language. An algorithm $\mathcal{A}$ is said *c-competitive* on L (with the constant K), if $\mathcal{A}(w) \leq c\,\mathrm{opt}(w) + K$ holds for each $w \in L$, where K depends on only $\mathcal{A}$ and L. Clearly, if an algorithm $\mathcal{A}$ is c-competitive on a language L, then it is c-competitive on any sublanguage $L' \subseteq L$, too. Hence, if an algorithm is efficient on V^*, then it is efficient on any language over V. Obviously, for any alphabet V, there exists a 1-competitive off-line algorithm on V^*.

We note that it has been proved in [CKPV] that there exists an off-line algorithm, which gives the optimal satisfaction of any word $w \in V^*$ in $O(k|w|^2)$ time. Hence, we can conclude that, for an arbitrary language L, there exists a polynomial 1-competitive off-line algorithm on L.

The class of on-line algorithms has not so nice property. Up to this time, the known best competitive ratio of any on-line algorithm on V^* is $2k-1$ (see [KP]). Moreover, a lower bound was presented in [MMS], which shows that the competitive ratio of any on-line algorithm on V^* is at least k. The question naturally arises that is it possible to reduce the competitive factor of on-line algorithms, if we consider only a special class of languages (e.g. $ONREG_0$)? Recall that $V^* \in REG$ holds, for any alphabet V, hence REG is not suitable for this purpose.

If it is not defined explicitly, by an algorithm we mean an on-line one in the rest of the paper.

A language $L \subseteq V^*$ is said *c-competitive*, if, for every M and k, there exists an algorithm, which is c-competitive on it. The 1-competitive languages are called *on-line languages*. The class of all on-line languages is denoted by $ONLINE$.

Lemma 2.7 *Every finite language is on-line.*

Proof. Suppose $L = \{w_1, \ldots, w_m\}$ is a finite language. We put $l = \max_{1 \leq i \leq m} |w_i|$. Let $\mathcal{A}$ be an arbitrary algorithm for L. Clearly, for any $w \in L$, $\mathcal{A}(w) \leq opt(w) + l\delta_{max}$. □

The properties of prefix words and prefix languages concerning the server problem are cornerstones in our proofs. Let L be an on-line language and let the algorithm $\mathcal{A}$ be 1-competitive on L. Suppose that $w \in L$ and $w = w_1 w_2$. Then, by (2) of Lemma 2.5, $\mathcal{A}(w) = \mathcal{A}(w_1 w_2) \leq \text{opt}(w) + K \leq \text{opt}(w_1) + \text{opt}(w_2) + k\delta_{max} + K$. Now let $S = S_1 S_2$ be the satisfaction of w given by $\mathcal{A}$, where S_1 is the satisfaction of the prefix w_1 and S_2 is the satisfaction of the suffix w_2. Clearly, $|S_1| \geq \text{opt}(w_1)$ and $k\delta_{max} + |S_2| \geq \text{opt}(w_2)$. Hence the following statements hold.

Observation 2.8 *If $\mathcal{A}$ is a 1-competitive algorithm on a language L with the constant K, $w \in L$ and $w = w_1 w_2$, then*
(1) $\mathcal{A}(w_1) \leq opt(w_1) + 2k\delta_{max} + K$ *and*
(2) the cost of $\mathcal{A}$ on the suffix w_2 is no more than $opt(w_2) + k\delta_{max} + K$.

By (1) of Observation 2.8, we have the following result immediately.

Theorem 2.9 *If $\mathcal{A}$ is a 1-competitive algorithm on a language L with the constant K, then $\mathcal{A}$ is 1-competitive on $L^{\sqsubseteq}$ with the constant $2k\delta_{max} + K$.*

3 The languages in $ONREG_0$ are on-line

The name $ONREG_0$ refers to the property of this class that it consists of on-line regular languages. However, it does not contain all on-line regular language.

In this section we prove that the languages in $ONREG_0$ are on-line. We do this by executing structural induction on the rules of Definition 2.3.

As the basic step of the structural induction, we prove that the languages $\{w\}$ and $\{w\}^*$ are on-line, for any word w (see (1) of Definition 2.3).

Lemma 3.1 *Let w be an arbitrary word over an arbitrary alphabet. Then $\{w\}$ and $\{w\}^*$ are on-line languages.*

Proof. As for the on-lineness of the language $\{w\}$, it immediately follows from Lemma 2.7. However, it also implied by the on-lineness of $\{w\}^*$, since $\{w\} \subseteq \{w\}^*$.

We construct a 1-competitive algorithm $\mathcal{A}$ on $\{w\}^*$. Informally speaking, $\mathcal{A}$ works as follows. We find a satisfaction $\underline{S}$ on an appropriate w-sequence, which has minimal cost density and results the same configuration as the starting one. Then $\mathcal{A}$ applies $\underline{S}$ repeatedly on any w-sequence.

We need some preparations. A satisfaction S of a request word w^{m_S} ($m_S \geq 1$) is called *circular* on w, if $\sigma_S = \overline{\sigma}_S$ holds. Denote the set of all circular satisfactions on w by $CSAT_w$. Moreover, a satisfaction S of a word w^{m_S} ($m_S \geq 0$) is called *noncircular* on w, if either $m_S = 0$, or $m_S \geq 1$ and $S = S_1 \dots S_{m_S}$, where each S_i is a satisfaction of the word w and $\sigma_{S_i} \neq \overline{\sigma}_{S_j}$, for any $1 \leq i \leq j \leq l$. Denote the set of all noncircular satisfactions on w by $NCSAT_w$. Observe that, since the number of different configurations is $\binom{n}{k}$, $m_S < \binom{n}{k}$ should hold for any noncircular satisfaction S.

Let the satisfaction $\underline{S} \in CSAT_w$ be such that $\frac{|\underline{S}|}{m_{\underline{S}}}$ is minimal in $CSAT_w$ and $m_{\underline{S}}$ is minimal with respect to the subset of $CSAT_w$ determined by the previous condition. Roughly speaking, $\underline{S}$ is one of the shortest satisfactions in $CSAT_w$, which has the minimal cost density.

We show that $\underline{S}$ exists and it can be computed. We prove this by showing that it is enough to consider only such kind of satisfactions S, for which $m_S \leq \binom{n}{k} - 1$ holds. Suppose that $m_{\underline{S}} > \binom{n}{k} - 1$. Decompose $\underline{S}$ as $\underline{S} = S_1 \dots S_{m_{\underline{S}}}$, where each S_i is a satisfaction of the word w, for any $1 \leq i \leq m_{\underline{S}}$. Clearly, $\sigma_{\underline{S}} = \sigma_{S_1} = \overline{\sigma}_{\underline{S}} = \overline{\sigma}_{S_{m_{\underline{S}}}}$ and $\overline{\sigma}_{S_i} = \sigma_{S_{i+1}}$. The number of different configurations is $\binom{n}{k}$, hence at least one of the following cases holds.

(i) There is an integer i, where $1 < i \leq m_{\underline{S}}$, such that $\sigma_{S_i} = \sigma_{\underline{S}}$. Let $S'_1 = S_1 \dots S_{i-1}$ and $S'_2 = S_i \dots S_{m_{\underline{S}}}$. Then $\underline{S} = S'_1 S'_2$ and $S'_1, S'_2 \in CSAT_w$ hold. Clearly, $m_{\underline{S}} = m_{S'_1} + m_{S'_2}$ and $|\underline{S}| = |S'_1| + |S'_2|$. Hence, either $\frac{|S'_1|}{m_{S'_1}} \leq \frac{|\underline{S}|}{m_{\underline{S}}}$ or $\frac{|S'_2|}{m_{S'_2}} \leq \frac{|\underline{S}|}{m_{\underline{S}}}$, which contradicts the minimality of $m_{\underline{S}}$.

(ii) There are integers i and j, where $1 < i < j < m_{\underline{S}}$, such that $\sigma_{S_i} = \sigma_{S_j}$. Let $S'_1 = S_1 \dots S_{i-1}$, $S'_2 = S_i \dots S_{j-1}$ and $S'_3 = S_j \dots S_{m_{\underline{S}}}$. Then $\underline{S} = S'_1 S'_2 S'_3$ and $S'_1 S'_3, S'_2 \in CSAT_w$ hold. Clearly, $m_{\underline{S}} = m_{S'_1 S'_3} + m_{S'_2}$ and $|\underline{S}| = |S'_1 S'_3| + |S'_2|$. It is easy to see that either $\frac{|S'_1 S'_3|}{m_{S'_1 S'_3}} \leq \frac{|\underline{S}|}{m_{\underline{S}}}$ or $\frac{|S'_2|}{m_{S'_2}} \leq \frac{|\underline{S}|}{m_{\underline{S}}}$, which contradicts the minimality of $m_{\underline{S}}$.

We have $m_{\underline{S}} \leq \binom{n}{k} - 1$. The subset of $CSAT_w$, which consists of the satisfactions S obeying $m_S \leq \binom{n}{k} - 1$, is finite, hence $\underline{S}$ can be computed.

Let now the algorithm $\mathcal{A}$ work as follows. Starting the satisfaction of a request word in $\{w\}^*$, $\mathcal{A}$ changes the starting configuration to $\sigma_{\underline{S}}$, then applies the satisfaction $\underline{S}$ repeatedly to the end of the input word. Clearly, $\mathcal{A}$ is on-line on $\{w\}^*$.

We prove that $\mathcal{A}$ is 1-competitive on $\{w\}^*$. Suppose that the input word is w^m, where $m \geq 0$. Let p the smallest integer such that $p \geq \frac{m}{m_{\underline{S}}}$. Then, by the definition of $\mathcal{A}$,

$$\mathcal{A}(w^m) \leq k\delta_{max} + p|\underline{S}| \qquad (*)$$

holds. Denote by S an optimal satisfaction of w^m in the given starting configuration, thus $|S| = opt(w^m)$. It is an easy exercise to show that the satisfaction

S can be decomposed as $S = S_0\hat{S}_1S_1 \ldots \hat{S}_lS_l$, where $\hat{S}_1, \hat{S}_2, \ldots, \hat{S}_l \in CSAT_w$ and $S_0S_1 \ldots S_l \in NCSAT_w$. By the property of noncircular satisfactions, we have $m_{S_0 \ldots S_l} < \binom{n}{k}$. Moreover, by the choice of $\underline{S}$, $\frac{|S|}{m_{\underline{S}}} \leq \frac{|\hat{S}_i|}{m_{\hat{S}_i}}$ holds, for each $1 \leq i \leq l$. Now we can calculate as follows.

$$\begin{aligned} \mathcal{A}(w^m) - \mathrm{opt}(w^m) &\leq p|\underline{S}| + k\delta_{max} - \textstyle\sum_{1 \leq i \leq l} |\hat{S}_i| \\ &\quad \text{(by } (*) \text{ and } |S| = \mathrm{opt}(w^m)\text{)} \\ &\leq k\delta_{max} + (pm_{\underline{S}} - \textstyle\sum_{1 \leq i \leq l} m_{\hat{S}_i}) \frac{|S|}{m_{\underline{S}}} \\ &\quad \text{(by } \tfrac{|S|}{m_{\underline{S}}} \leq \tfrac{|\hat{S}_i|}{m_{\hat{S}_i}}\text{)}. \end{aligned}$$

Moreover, since

$$\begin{aligned} pm_{\underline{S}} - \textstyle\sum_{1 \leq i \leq l} m_{\hat{S}_i} &\leq pm_{\underline{S}} - m + \binom{n}{k} \\ &\quad \text{(by } m = \textstyle\sum_{1 \leq i \leq l} m_{\hat{S}_i} + m_{S_0 \ldots S_l},\ m_{S_0 \ldots S_l} < \tbinom{n}{k}\text{)} \\ &\leq m_{\underline{S}} + \binom{n}{k} \\ &\quad \text{(by the choice of } p\text{)}, \end{aligned}$$

we have

$$\mathcal{A}(w^m) \leq \mathrm{opt}(w^m) + (k\delta_{max} + |\underline{S}| + \frac{|S|}{m_{\underline{S}}}\binom{n}{k}).$$

Hence $\mathcal{A}$ is 1-competitive on $\{w\}^*$. □

In the next step of the structural induction, we prove that the two construction rules defined in (2) of Definition 2.3 preserve the on-lineness.

Lemma 3.2 *For any alphabet V, language $L \in ONREG_0$ and word $w \in V^*$, if L is on-line, then $L\{w\}$ and $L\{w\}^*$ are also on-line languages.*

Proof. Since $L\{w\} \subseteq L\{w\}^*$ holds, it is enough to show that $L\{w\}^*$ is on-line to prove the lemma.

The language L is on-line, hence there is an algorithm $\mathcal{A}_1$, which is 1-competitive on it. Moreover, by Lemma 3.1, the language $\{w\}^*$ is on-line. Let $\mathcal{A}_2$ be a 1-competitive algorithm on $\{w\}^*$. We construct a 1-competitive algorithm $\mathcal{A}$ on $L\{w\}^*$. Informally, $\mathcal{A}$ applies $\mathcal{A}_1$ as long as possible, then finds the beginning of the remainder w-sequence (if there is) and applies $\mathcal{A}_2$ to the end of the request word.

Observe that every input word is of the form xw^m, where $x \in L$ and $m \geq 0$. Let $x \in L$ and $m \geq 1$ be arbitrary, but fixed in the sequel. (As for the case $m = 0$, the 1-competitiveness of $\mathcal{A}$ will follow from its construction immediately.) Observe that xw^m can be decomposed unambiguously as $xw^{m_1}w_1w_2w^{m_2}$, where $m = m_1 + m_2 + 1$, $w_1w_2 = w$ and $xw^{m_1}w_1$ is the longest prefix of xw^m, which is in $L^{\sqsubseteq}$.

Let the algorithm $\mathcal{A}$ work as follows.

1. While the scanned prefix of the input word xw^m is in $L^{\sqsubseteq}$, $\mathcal{A}$ applies $\mathcal{A}_1$. Observe that in this way $\mathcal{A}_1$ is applied exactly on $xw^{m_1}w_1$.

2. Then $\mathcal{A}$ reads the letters successing $xw^{m_1}w_1$, satisfies them arbitrarily and stores their concatenation, until the stored word is of the form yw or the

input ends. Since $|w_2| \leq |w|$, it should be clear that, for any input word, less than $2|w|$ letters are to be read in this step.

3. If there are no more letters left on the input, then $\mathcal{A}$ terminates. Otherwise, observe that $w_2 w^{m_2} = yww^{m_2-1}y'$ should hold, for some y, where $yy' = w_2$ and $y' \sqsubseteq w$. Moreover, the rest of the input is $w^{m_2-1}y'$, hence $\mathcal{A}_2$ can be applied on the remainder input without any difficulties. Let $\mathcal{A}$ apply $\mathcal{A}_2$ on the rest of the input.

Now we prove that $\mathcal{A}$ is 1-competitive on $L\{w\}^*$. Supposing $xw^m \in L^{\sqsubseteq}$ $(m \geq 0)$, by Lemma 2.9, $\mathcal{A}(xw^m) \leq \mathrm{opt}(xw^n) + K_1 + 2k\delta_{max}$ holds. Now suppose that $xw^m \notin L^{\sqsubseteq}$. Clearly, in this case $m \geq 1$ should hold. We can calculate as follows.

$$\begin{aligned}
\mathcal{A}(xw^n) &\leq \mathcal{A}_1(xw^{m_1}w_1) + 2|w|\delta_{max} + \mathcal{A}_2(w^{m_2}) \\
&\quad \text{(by the construction of } \mathcal{A}) \\
&\leq \mathrm{opt}(xw^{m_1}w_1) + K_1 + 2k\delta_{max} + \mathrm{opt}(w^{m_2}) + K_2 + 2|w|\delta_{max} \\
&\quad \text{(by Lemma 2.9)} \\
&\leq \mathrm{opt}(xw^{m_1}w_1w_2) + \mathrm{opt}(w^{m_2}) + K_1 + K_2 + 2(k + |w|)\delta_{max} \\
&\leq \mathrm{opt}(xw^{m_1}w_1w_2w^{m_2}) + k\delta_{max} + K_1 + K_2 + 2(k + |w|)\delta_{max} \\
&\quad \text{(by Lemma 2.5)} \\
&= \mathrm{opt}(xw^m) + (K_1 + K_2 + (3k + 2|w|)\delta_{max})
\end{aligned}$$

Hence $\mathcal{A}$ is 1-competitive on $L\{w\}^*$. □

Finally, we have to check the construction rule defined in (3) of Definition 2.3 to complete the proof.

Lemma 3.3 *For any languages $L_1, L_2 \in ONREG_0$, if both L_1 and L_2 are on-line then the language $L_1 \cup L_2$ is on-line, too.*

Proof. Since L_1 and L_2 are on-line languages, there exist algorithms $\mathcal{A}_1$ and $\mathcal{A}_2$ such that, for any words $w_1 \in L_1$ and $w_2 \in L_2$, $\mathcal{A}_1(w_1) \leq \mathrm{opt}(w_1) + K_1$ and $\mathcal{A}_2(w_2) \leq \mathrm{opt}(w_2) + K_2$ hold, where K_1 and K_2 are constants. We construct an algorithm $\mathcal{A}$, which is 1-competitive on $L_1 \cup L_2$.

Let the input word be an arbitrary $w \in L_1 \cup L_2$. The algorithm $\mathcal{A}$ works as follows.

1. Reading the next letter of w, on the basis of the scanned prefix $w' \sqsubseteq w$, $\mathcal{A}$ tries to decide that w which of the languages L_1 and L_2 belongs to. Recall that both $w' \in L_1^{\sqsubseteq}$ or $w' \in L_2^{\sqsubseteq}$ are decidable. While this cannot be decided unambiguously, $\mathcal{A}$ computes the satisfactions and the new configurations determined by both $\mathcal{A}_1$ and $\mathcal{A}_2$, but satisfies the request as suggested by $\mathcal{A}_1$. However, it stores the configuration computed by $\mathcal{A}_2$.

2. If $\mathcal{A}$ detects that w cannot be in L_2, then it finishes the satisfaction of w continuing by $\mathcal{A}_1$. Otherwise, if it turns out that w is not in L_1, then $\mathcal{A}$ changes the configuration to the one stored for $\mathcal{A}_2$, and completes the satisfaction of w by $\mathcal{A}_2$.

We show that it is 1-competitive on $L_1 \cup L_2$. Observe that, for any input word $w \in L_1 \cup L_2$, exactly one of the following two cases holds.

(i) $w \in L_1^{\sqsubseteq}$. By the construction of $\mathcal{A}$, in this case $\mathcal{A}(w) = \mathcal{A}_1(w)$ holds. Since $\mathcal{A}_1$ is 1-competitive on L_1, by Proposition 2.9 we have $\mathcal{A}(w) \leq \mathrm{opt}(w) + K_1 + 2k\delta_{max}$, where K_1' is a constant.

(ii) $w \in (L_2 - L_1^{\sqsubseteq})$. In this case $\mathcal{A}$ works as follows. The word w can be decomposed unambiguously as $w = w_1 w_2$, where w_1 is the longest prefix of w such that $w_1 \in L_1^{\sqsubseteq}$. The algorithm $\mathcal{A}$ implements $\mathcal{A}_1$ on w_1, changes the configuration and implements $\mathcal{A}_2$ on w_2. Recall that changing the configuration costs at most $k\delta_{max}$. We can calculate as follows.

$$\begin{aligned}
\mathcal{A}(w) &\leq \mathcal{A}_1(w_1) + k\delta_{max} + \mathrm{opt}(w_2) + K_2 + k\delta_{max} \\
&\quad \text{(by (2) of Observation 2.8)} \\
&\leq \mathrm{opt}(w_1) + K_1 + 2k\delta_{max} + \mathrm{opt}(w_2) + K_2 + 2k\delta_{max} \\
&\quad \text{(by Lemma 2.9)} \\
&\leq \mathrm{opt}(w) + (K_1 + K_2 + 4k\delta_{max}) \\
&\quad \text{(by (1) of Lemma 2.5)}
\end{aligned}$$

We have that $\mathcal{A}$ is 1-competitive on $L_1 \cup L_2$. □

With this, we are ready to prove the main result of our paper.

Theorem 3.4 *Every language in $ONREG_0$ is on-line. Moreover, given a language $L \in ONREG_0$, an algorithm can be constructed effectively, which is 1-competitive on L.*

Proof. Recall that every language $L \in ONREG_0$ can be constructed as described in Definition 2.3. Hence, by the lemmas 3.1, 3.2, 3.3 and by the principline of structural induction, we have that $L \in ONLINE$ holds. Moreover, by the application of the constructions in the proofs of the above lemmas, a 1-competitive algorithm can be given for L. □

Recall that $\mathrm{opt}(w) = \mathrm{opt}(\mathrm{lr}(w))$ holds, for any word w. Moreover, the competitive ratio of any algorithm on any sublanguage of a given language is no more, than its competitive ratio on the whole language. By these observations, we can show the on-lineness of certain other languages, which can be even not regular.

Corollary 3.5 *For an arbitrary language L, if there exists a language $L' \in ONREG_0$ such that $L \subseteq L'$ or $\mathrm{lr}(L) \subseteq L'$ hold, then L is on-line. Moreover, in this case a 1-competitive algorithm can be constructed effectively for L.*

Finally, we present three examples for the application of Corollary 3.5.

(1) The language $L_1 = \{a^n b^n c^n \mid n \geq 0\}$ is a well known nonregular language. However, by Corollary 3.5, it is on-line, since $\mathrm{lr}(L_1) = \{e, abc\} \in ONREG_0$.

(2) Consider $L_2 = \{w_1^n w_2^n \mid n \geq 0\}$, where w_1 and w_2 are arbitrary words. Note that generally L_2 is not regular. However, $L_2 \in ONLINE$ follows from the fact that L_2 is a subset of the language $(w_1)^*(w_2)^*$, which is in $ONREG_0$.

(3) It can be shown that $L_3 = (a \cup abc)^*$ is not in $ONREG_0$, but it is regular. Observe that L_3 can be defined by $(a^*abc)^*a^*$ as well. Since $\mathrm{lr}((a^*abc)^*a^*) \subseteq (abc)^*a^* \in ONREG_0$, we have that L_3 is on-line language.

4 Related results

The question obviously arises that if there exists an on-line language L, of which the on-lineness cannot be proved by Corollary 3.5? (That is, such L that $\mathrm{lr}(L)$ is not a subset of any language in $ONREG_0$.) Specially, is there such kind of language in REG? These problems are open up to this time. However, we have the following results, which shows that if there exists a language in REG with the above property, then it should be very special one.

Recall that the construction of a language in $ONREG_0$ differs from the construction of a general regular one in the point that the closure operation is allowed only for singleton languages. Moreover, by Theorem 3.4, every language in $ONREG_0$ is necessarily on-line. Hence one can guess that, roughly speaking, a regular language may loose the on-lineness, when the closure is applied for a multielement set during its construction. The following lemma shows that this really holds in most cases.

Theorem 4.1 *Let V be an arbitrary alphabet with $|V| \geq 3$. Consider any two words $w_1, w_2 \in V^*$ such that w_1 contains at least two different letters and there is a letter in w_2, which does not occur in w_1. Then the language $(w_1 \cup w_2)^*$ is not on-line.*

Proof. Assume that the number of servers (i.e. k) is 2. It is sufficient to show that there exists a metric space $M = (V, \delta)$, in which the competitive ratio of L is greater than 1.

Denote by a the letter, which occurs in w_2 and not contained by w_1. For any different letters $u, v \in V - \{a\}$, let $\delta(u, v) = 1$ and, for each $v \in V - \{a\}$, let $\delta(v, a) = D$, where D is defined later. Suppose that the starting configuration always contains a server on a.

Let us consider a request-answer game, where a requester R plays against an algorithm $\mathcal{A}$. In each round, R gives a request, which is satisfied by $\mathcal{A}$ immediately. We show that defining an appropriately large D, for any algorithm $\mathcal{A}$, R has a strategy providing that the difference of the costs of the satisfaction generated by $\mathcal{A}$ and the optimal one grows unboundedly. This implies that the competitive ratio of L is greater than 1.

Let $H = \{w_2 w_1 w_2, \ldots, w_2 w_1^t w_2\}$, where $t > 1$ is defined later. We assume that the request words composed by R are chosen from H^*. Since $H^* \subseteq L$, to prove the

lemma, it is enough to show that H^* is not on-line. Observe that any word $w \in H^*$ consists of sections of the form $w_2 w_1^s w_2$, where $1 \leq s \leq t$.

Let R compose its request words dynamically, sections by sections, obeying the following rules.

Rule 1 If, for some $1 \leq s < t$, there is a server on a, when $w_2 w_1^{s-1}$ is satisfied, and $\mathcal{A}$ moves that server processing the next (sth) w_1, then let this section be $w_2 w_1^s w_2$. Specially, if $\mathcal{A}$ moves the server from a while satisfying the initial w_2, let R chose $w_2 w_1 w_2$.

Rule 2 If $\mathcal{A}$ does not move the server from a while processing the w_1-s, then let R choose $w_2 w_1^t w_2$.

We need some technical preparations. Since w_1 contains at least two different letters, it should be clear that $|\mathrm{lr}(w_1)| \geq 2$. Moreover, for any w_1, one of the following cases holds, where $s \geq 1$.

Case 1 $\mathrm{first}(w_1) \neq \mathrm{last}(w_1)$. Then $|\mathrm{lr}(w_1^s)| = s|\mathrm{lr}(w_1)|$.

Case 2 $\mathrm{first}(w_1) = \mathrm{last}(w_1)$. Then $|\mathrm{lr}(w_1^s)| = s(|\mathrm{lr}(w_1)| - 1) + 1$.

Suppose that a section is chosen by Rule 1. Then the cost of the satisfaction by $\mathcal{A}$ is at least $(s-1)|\mathrm{lr}(w_1)| + D$ in Case 1, and $(s-1)(|\mathrm{lr}(w_1)| - 1) + 1 + D$ in Case 2. However, if we do not move the server on a, then this section could be satisfied with cost no more than $|\mathrm{lr}(w_2)| + s|\mathrm{lr}(w_1)| + |\mathrm{lr}(w_2)|$ in Case 1, and $|\mathrm{lr}(w_2)| + s(|\mathrm{lr}(w_1)| - 1) + 1 + |\mathrm{lr}(w_2)|$ in Case 2. Hence, if

$$D > |\mathrm{lr}(w_1)| + 2|\mathrm{lr}(w_2)|$$

holds, then $\mathcal{A}$ is more expensive on this section than the our one.

Now suppose that the section is determined by Rule 2, that is $\mathcal{A}$ does not move the server from a during the processing of w_1-s. Then the cost of $\mathcal{A}$ is at least $t|\mathrm{lr}(w_1)|$ in Case 1, and $t(|\mathrm{lr}(w_1)| - 1) + 1$ in Case 2. However, if we move the server from a to $\mathrm{first}(w_1)$ after the initial w_2, leave there while processing the w_1-s, and move back to a before the final w_2, then this section costs at most $|\mathrm{lr}(w_2)| + D + t(|\mathrm{lr}(w_1)| - 1) + D + |\mathrm{lr}(w_2)|$ in Case 1, and $|\mathrm{lr}(w_2)| + D + t(|\mathrm{lr}(w_1)| - 2) + D + |\mathrm{lr}(w_2)|$ in Case 2. Therefore, if

$$t > 2|\mathrm{lr}(w_2)| + 2D$$

holds, then $\mathcal{A}$ is more expensive than the our one.

For any words w_1 and w_2, the values of D and t can be computed and fixed as above. We have that, for an arbitrary on-line algorithm $\mathcal{A}$, R has a strategy, which proves that $\mathcal{A}$ costs more than an off-line algorithm. Since an input word $w \in H^*$ can contain arbitrary many sections, this difference grows unboundedly, hence $\mathcal{A}$ is not 1-competitive. This implies that H^* (and hence L) is not on-line language. □

Generally, a regular language is defined by a regular expression. The following result shows that if a subexpression defines a not on-line language, then the language defined by the whole expression cannot be on-line.

Theorem 4.2 *Let the language L defined by the regular expression E. Consider any subexpression E' of E. If the language defined by E' is not on-line then L is not on-line, too.*

Proof. It is a routine exercise to prove the theorem using structural induction on the defining rules of *REG*. □

We show two examples for the application of theorems 4.1 and 4.2.

(1) The language $(a \cup bc)^*$ is not on-line by Theorem 4.1.

(2) Consider the regular expression $((abc \cup bcd)^*d \cup abcd)^*$. Its subexpression $(abc \cup bcd)^*$ defines a not on-line language by Theorem 4.1, hence, by Theorem 4.2, the language defined by the whole expression is not on-line, too.

Finally, we summarize some abstract properties of the language class $ONLINE$ in the following theorem.

Theorem 4.3 *Let L_1 and L_2 be arbitrary on-line languages, then*

(1) for any $L \subseteq L_1$, L is on-line,
(2) $L_1^{\sqsubseteq}$ is on-line,
(3) $L_1 \cap L_2$ is on-line,
(4) if the prefix problem is decidable both for L_1 and L_2, then $L_1 \cup L_2$ is on-line,
(5) L_1^* is generally not on-line,
(6) $\overline{L_1}$ is generally not on-line.

Proof. The statements (1) and (2) have been proved earlier in this paper. Moreover, (3) follows from (1) immediately. We can get (4) by slightly modifying the proof of Lemma 3.3. The statement (5) can be proved by Theorem 4.1. For proving (6), let us assume, that $L_1 = \{w\}^*$, and let u be a letter, which is different from the last letter of w. Let w_1 and w_2 be any words satisfying the conditions in Theorem 4.1. Then we have that $(w_1u \cup w_2u)^* \subseteq \overline{L_1}$. Thus, $\overline{L_1}$ is not on-line language. □

References

[BIRS] Borodin, A., Irani, S., Raghavan, P., Schieber, B., *Competitive Paging with Locality of Reference*, STOC 91, pp. 249-259

[CKPV] Chrobak, M., Karloff, H., Payne, T. and Vishwanathan, S., *New Results on Server Problems*, SIAM J. Disc. Math., Vol. 4, No. 2, May 1991, pp. 172-181

[FK] Fiat, A., Karlin, A., *Randomized and Multipointer Paging with Locality of Reference*, STOC 95, pp. 626-634

[KP] Koutsoupias, E. and Papadimitrou, C., *On the k-Server Conjecture*, STOC 94, pp. 507-511

[HU] Hopcroft, J. E. and Ullman, J. D., *Introduction to Automata Theory, Languages and Computation*, Addison-Wesley, Series in Computer Science, 1979

[MMS] Manasse, M. S., McGeoch L. A. and Sleator, D. D., *Competitive Algorithms for Server Problems*, Journal of Algorithms 11 (1990), pp. 208-230

[ST] Sleator, D. D., Tarjan, R. E., *Amortized Efficiency of List Update and Paging Rules*, Comm. of the ACM, February 1985, pp. 202-208

[W] Winskel, G., *The Formal Semantics of Programming Languages, An Introduction*, The MIT Press, Foundations of Computing Series, 1993

Received May, 1996

Acta Cybernetica **13** (1997) 41-54.

On α_i-products of nondeterministic tree automata*

B. Imreh†

Abstract

In this paper, we characterize the isomorphically complete systems of nondeterministic tree automata with respect to the family of α_i-products. In particular, our characterization yields that any finite nondeterministic tree automata can be embedded isomorphically into a suitable serial product of two-state nondeterministic tree automata.

Keywords: nondeterministic tree automata, composition, completeness

1 Introduction

Isomorphic representation of automata by different compositions is one of the central problems in the theory of automata. One line of the researches is to characterize those systems of automata which are isomorphically complete, i.e., every automaton is an isomorphic image of a subautomaton of a product from them. Most of the studies regarding characterizations of isomorphically complete systems concern deterministic automata or deterministic tree automata. We quote only [1],[3],[4],[7],[9],[10],[11], and [15]. On the other hand, together with the spread of parallel computation, the importance of nondeterministic automata is increasing. This is the motivation to deal with the representations of nondeterministic automata. The first description of the isomorphically complete systems of nondeterministic automata with respect to the general product was given in [5]. In the work [6], it is proved that the cube-product is equivalent to the general product regarding isomorphically complete systems of nondeterministic automata. The isomorphic representation of a special class of nondeterministic automata is investigated in [12]. The notion of α_i-product (cf. [2],[3]) was extended to nondeterministic automata, and the isomorphically complete systems were characterized with respect to this hierarchy of products in [14]. From this characterization, it turns out that contrary to the deterministic case, in the nondeterministic case, there exist finite isomorphically complete systems with respect to the α_0-product, furthermore, the α_i-product is equivalent to the general product regarding isomorphically complete systems if $i \geq 1$. The isomorphically complete systems of nondeterministic tree automata

*This work has been supported by the the Hungarian National Foundation for Scientific Research, Grant 014888 and and the Ministry of Culture and Education of Hungary, Grant 223/95.

†Dept. of Informatics, József Attila University, Árpád tér 2, H-6720 Szeged, Hungary

with respect to the general product and the cube-product are studied in [13] where it is proved that these compositions are equivalent regarding isomorphically complete systems. Here, using the characterization presented in [13] and extending the notion of α_i-product to nondeterministic tree automata, we generalize the result of [14] for nondeterministic tree automata. Namely, we prove that there exist finite isomorphically complete systems of nondeterministic tree automata with respect to the α_0-product, moreover, the α_i-product is equivalent to the general product regarding isomorphically complete systems of nondeterministic tree automata if $i \geq 1$.

The paper is organized as follows. In Section 2, the necessary notions and notations are introduced. The following part, Section 3, presents the characterization of the isomorphically complete systems of nondeterministic tree automata with respect to the α_0-product. Finally, Section 4 is devoted to the description of the isomorphically complete systems of nondeterministic tree automata regarding α_i-product with $i \geq 1$.

2 Preliminaries

To start the discussion, we introduce some notions and notations of relational systems (cf. [8]). By a set of *relational symbols*, we mean a nonempty union $\Sigma = \Sigma_1 \cup \Sigma_2 \cup \dots$ where Σ_m, $m = 1, 2, \dots$, are pairwise disjoint sets of symbols. For any $m \geq 1$, the set Σ_m is called the set of *m-ary relational symbols*. It is said that the *rank* or *arity* of a symbol $\sigma \in \Sigma$ is m if $\sigma \in \Sigma_m$. Now, let a set Σ of relational symbols and a set R of positive integers be given. R is called the *rank-type* of Σ if, for any integer $m \geq 0$, $\Sigma_m \neq \emptyset$ if and only if $m \in R$. In the sequel, we shall work under a fixed rank-type R.

Now, let Σ be a set of relational symbols with rank-type R. By a *nondeterministic Σ-algebra* $\mathcal{A}$, we mean a pair consisting of a nonempty set A and a mapping that assigns to every relational symbol $\sigma \in \Sigma$ an m-ary relation $\sigma^{\mathcal{A}} \subseteq A^m$ where the arity of σ is m. The set A is called the *set of elements of* $\mathcal{A}$ and $\sigma^{\mathcal{A}}$ is the *realization of σ in $\mathcal{A}$*. The mapping $\sigma \to \sigma^{\mathcal{A}}$ will not be mentioned explicitly, we only write $\mathcal{A} = (A, \Sigma)$. For every $m \in R$, $\sigma \in \Sigma_m$, and $(a_1, \dots, a_{m-1}) \in A^{m-1}$, we denote the set $\{a : a \in A \ \& \ \sigma^{\mathcal{A}}(a_1, \dots, a_{m-1}, a)\}$ by $(a_1, \dots, a_{m-1})\sigma^{\mathcal{A}}$. If $(a_1, \dots, a_{m-1})\sigma^{\mathcal{A}}$ is a one-element set $\{a\}$, then we write $(a_1, \dots, a_{m-1})\sigma^{\mathcal{A}} = a$.

It is said that a nondeterministic Σ-algebra $\mathcal{A}$ is *finite* if A is finite, and it is of *finite type* if Σ is finite. By a *nondeterministic tree automaton*, we mean a finite nondeterministic Σ-algebra of finite type. Finally, it is said that the *rank-type of a nondeterministic tree automaton* $\mathcal{A} = (A, \Sigma)$ is R if the rank-type of Σ is R.

Let $\mathcal{A} = (A, \Sigma)$ and $\mathcal{B} = (B, \Sigma)$ be nondeterministic tree automata with rank-type R. $\mathcal{B}$ is called a *subautomaton* of $\mathcal{A}$ if $B \subseteq A$ and, for all $m \in R$ and $\sigma \in \Sigma_m$, $\sigma^{\mathcal{B}}$ is the restriction of $\sigma^{\mathcal{A}}$ to B^m. A one-to-one mapping μ of A onto B is called an *isomorphism* of $\mathcal{A}$ onto $\mathcal{B}$ if $\sigma^{\mathcal{A}}(a_1, \dots, a_m)$ if and only if $\sigma^{\mathcal{B}}(\mu(a_1), \dots, \mu(a_m))$,

for all $m \in R$, $(a_1, \ldots, a_m) \in A^m$, $\sigma \in \Sigma_m$. In this case, it is said that $\mathcal{A}$ and $\mathcal{B}$ are *isomorphic*. It is easy to see that μ is an isomorphism of $\mathcal{A}$ onto $\mathcal{B}$ if and only if $(a_1, \ldots, a_{m-1})\sigma^{\mathcal{A}}\mu = (\mu(a_1), \ldots, \mu(a_{m-1}))\sigma^{\mathcal{B}}$ holds, for all $m \in R$, $\sigma \in \Sigma_m$, $(a_1, \ldots, a_{m-1}) \in A^{m-1}$.

In the case of classical automata, a composition of automata can be visualized as a network in which each vertex denotes an automaton and the actual input sign of a component automaton may depend on the input sign of the whole composition and only on those automata which have a direct connection to the component automaton under consideration. From practical point of view, one of the most self-evident networks is the well-known serial or cascade connection. In this case, the composition can be considered as a chain in which each machine has a direct connection with all the previous ones. Generalizing this concept, F. Gécseg [2] introduced a family of compositions, the α_i-products, where i is a nonnegative integer which denotes the maximal admissible length of feedbacks. Now, we extend the notion of α_i-product to nondeterministic tree automata.

Let us denote the class of all nondeterministic tree automata with rank-type R by $\mathbf{U}_R$. In general, a composition of nondeterministic tree automata from $\mathbf{U}_R$ can be visualized as a network in which each vertex denotes a nondeterministic tree automaton in $\mathbf{U}_R$ and the actual relation of a component automaton may depend on the relational symbol of the whole composition and only on those nondeterministic tree automata which have a direct connection to the component under consideration. In particular, the formal definition of the α_i-product of nondeterministic tree automata can be given as follows.

Let i be an arbitrary nonnegative integer. Let us consider the nondeterministic tree automata $\mathcal{A} = (A, \Sigma) \in \mathbf{U}_R$ and $\mathcal{A}_j = (A_j, \Sigma^{(j)}) \in \mathbf{U}_R$, $j = 1, \ldots, n$. Furthermore, let us take a family Ψ of mappings

$$\Psi_{mj} : (A_1 \times \ldots \times A_{j+i-1})^{m-1} \times \Sigma_m \to \Sigma_m^{(j)}, \quad m \in R, \quad 1 \leq j \leq n .$$

It is said that $\mathcal{A}$ is the α_i-product of $\mathcal{A}_j$, $j = 1, \ldots, n$, with respect to Ψ if the following conditions are satisfied:

(i) $$A = \prod_{j=1}^{n} A_j ,$$

(ii) for any $m \in R$, $\sigma \in \Sigma_m$ and $((a_{1,1}, \ldots, a_{1,n}), \ldots, (a_{m-1,1}, \ldots, a_{m-1,n})) \in A^{m-1}$,

$$((a_{1,1}, \ldots, a_{1,n}), \ldots, (a_{m-1,1}, \ldots, a_{m-1,n}))\sigma^{\mathcal{A}} =$$

$$(a_{1,1}, \ldots, a_{m-1,1})\sigma_1^{\mathcal{A}_1} \times \ldots \times (a_{1,n}, \ldots, a_{m-1,n})\sigma_n^{\mathcal{A}_n} ,$$

where

$$\sigma_j = \Psi_{mj}((a_{1,1}, \ldots, a_{1,j+i-1}), \ldots, (a_{m-1,1}, \ldots, a_{m-1,i+j-1}), \sigma), \qquad j = 1, \ldots, n\,.$$

We shall use the notation

$$\prod_{j=1}^{n} \mathcal{A}_j(\Sigma, \Psi)$$

for the product introduced above. In particular, if $\mathcal{A}_j$, $j = 1, \ldots, n$, are identical copies of some nondeterministic tree automaton $\mathcal{B}$, then we speak of an *α_i-power* and we write $\mathcal{B}^n(\Sigma, \Psi)$ for $\prod_{j=1}^{n} \mathcal{A}_j(\Sigma, \Psi)$.

Let $\mathbf{B}$ be a system of nondeterministic tree automata from $\mathbf{U}_R$. It is said that $\mathbf{B}$ is *isomorphically complete for $\mathbf{U}_R$ with respect to the α_i-product* if any nondeterministic tree automaton from $\mathbf{U}_R$ is isomorphic to a subautomaton of an α_i-product of nondeterministic tree automata in $\mathbf{B}$.

3 α_0-product

In this section, we deal with the first member of this family of products, the α_0-product, which correspondes to the serial composition. In this case, the feedback functions can be given as follows:

$$\Psi_{1j} : \Sigma_1 \to \Sigma_1^{(j)},\ j = 1, \ldots, n, \ \text{ if } \ 1 \in R,$$

$$\Psi_{m1} : \Sigma_m \to \Sigma_m^{(1)}, \ \ 1 \neq m \in R,$$

$$\Psi_{mj} : (A_1 \times \ldots \times A_{j-1})^{m-1} \times \Sigma_m \to \Sigma_m^{(j)}, \quad 1 \neq m \in R, \ \ 2 \leq j \leq n\,.$$

In what follows, we need a special two-state nondeterministic tree automaton which is defined in the following way. For all $m \in R$, let us assign a symbol to each m-ary relation on $\{0, 1\}$. Let $\bar{\Sigma}_m$ denote the set of these relational symbols and let $\bar{\Sigma} = \cup_{m \in R} \bar{\Sigma}_m$. Let us define the nondeterministic tree automaton $\mathcal{G} = (\{0, 1\}, \bar{\Sigma})$ such that, for every $m \in R$ and $\sigma \in \bar{\Sigma}_m$, $\sigma^{\mathcal{G}}$ is the corresponding m-ary relation on $\{0, 1\}$.

The following theorem provides necessary and sufficient conditions for a system of nondeterministic tree automata from $\mathbf{U}_R$ to be isomorphically complete for $\mathbf{U}_R$ with respect to the α_0-product.

Theorem 1. *A system $\mathbf{B}$ of nondeterministic tree automata from $\mathbf{U}_R$ is isomorphically complete for $\mathbf{U}_R$ with respect to the α_0-product if and only if*

(a) *there exists a nondeterministic tree automaton* $\mathcal{A}^* = (A^*, \Sigma^*) \in \mathbf{B}$ *such that* $\mathcal{A}^*$ *has two different elements* a_0^*, a_1^*, *and for every* $1 \neq m \in R$, *there is a* $\sigma_m \in \Sigma_m^*$ *for which* $(a_{s_1}^*, \ldots, a_{s_{m-1}}^*)\sigma_m^{\mathcal{A}^*} \supseteq \{a_0^*, a_1^*\}$ *is valid, for all* $(s_1, \ldots s_{m-1}) \in \{0,1\}^{m-1}$, *furthermore, there is a* $\sigma_1 \in \Sigma_1^*$ *with* $\{a_0^*, a_1^*\} \subseteq \sigma_1^{\mathcal{A}^*}$ *if* $1 \in R$,

(b) *for all* $m \in R$ *and* $\mathbf{i} = (i_1, \ldots, i_m) \in \{0,1\}^m$, $\mathbf{B}$ *contains a nondeterministic tree automaton* $\mathcal{A}^{(\mathbf{i})} = (A^{(\mathbf{i})}, \Sigma^{(\mathbf{i})})$ *satisfying the following conditions:*

(b1) $A^{(\mathbf{i})}$ *has two different elements* $a_0^{(\mathbf{i})}$ *and* $a_1^{(\mathbf{i})}$,

(b2) *there exists a* $\sigma_{\mathbf{i}} \in \Sigma_m^{(\mathbf{i})}$ *with* $(a_{i_1}^{(\mathbf{i})}, \ldots, a_{i_{m-1}}^{(\mathbf{i})})\sigma_{\mathbf{i}}^{\mathcal{A}^{(\mathbf{i})}} \cap \{a_0^{(\mathbf{i})}, a_1^{(\mathbf{i})}\} = \{a_{i_m}^{(\mathbf{i})}\}$,

(b3) *for all* $1 \neq u \in R$ *and* $\mathbf{s} = (s_1, \ldots, s_{u-1}) \in \{0,1\}^{u-1}$, *there is a* $\sigma_{\mathbf{i},\mathbf{s}} \in \Sigma_u^{(\mathbf{i})}$ *for which* $\{a_0^{(\mathbf{i})}, a_1^{(\mathbf{i})}\} \subseteq (a_{s_1}^{(\mathbf{i})}, \ldots, a_{s_{u-1}}^{(\mathbf{i})})\sigma_{\mathbf{i},\mathbf{s}}^{\mathcal{A}^{(\mathbf{i})}}$, *furthermore, there is a* $\bar{\sigma}_{\mathbf{i}} \in \Sigma_1^{(\mathbf{i})}$ *with* $\{a_0^{(\mathbf{i})}, a_1^{(\mathbf{i})}\} \subseteq \bar{\sigma}_{\mathbf{i}}^{\mathcal{A}^{(\mathbf{i})}}$ *if* $1 \in R$.

Proof. To prove the necessity, let us suppose that $\mathbf{B}$ is an isomorphically complete system of nondeterministic tree automata for $\mathbf{U}_R$ with respect to the α_0-product. Then there are $\mathcal{A}_j = (A_j, \Sigma^{(j)}) \in \mathbf{B}$, $j = 1, \ldots, n$, such that $\mathcal{G}$ is isomorphic to a subautomaton $\mathcal{A} = (A, \bar{\Sigma})$ of an α_0-product $\prod_{j=1}^n \mathcal{A}_j(\bar{\Sigma}, \Psi)$. Let μ denote a suitable isomorphism and let

$$\mu(0) = (a_{0,1}, \ldots, a_{0,n}) \text{ and } \mu(1) = (a_{1,1}, \ldots, a_{1,n}).$$

Let us denote by k the smallest index with $a_{0,k} \neq a_{1,k}$. Then we prove that $\mathcal{A}_k$ satisfies condition (a). For this purpose, we distinguish two cases depending on m.

Let us suppose that $m \neq 1$. By the definition of $\mathcal{G}$, each m-ary relation on $\{0,1\}$ has a relational symbol in $\bar{\Sigma}_m$. Thus, there exists a $\bar{\sigma}_m \in \bar{\Sigma}_m$ such that $\bar{\sigma}_m^{\mathcal{G}}$ is the complete m-ary relation on $\{0,1\}$. This means that $\bar{\sigma}_m^{\mathcal{G}}(s_1, \ldots, s_m)$ is valid, for all $(s_1, \ldots, s_m) \in \{0,1\}^m$. Therefore, $(s_1, \ldots, s_{m-1})\bar{\sigma}_m^{\mathcal{G}} = \{0,1\}$, and thus, $(s_1, \ldots, s_{m-1})\bar{\sigma}_m^{\mathcal{G}}\mu = \{0,1\}\mu = \{\mu(0), \mu(1)\}$ is valid, for all $(s_1, \ldots, s_{m-1}) \in \{0,1\}^{m-1}$. Since μ is an isomorphism, we have $(s_1, \ldots, s_{m-1})\bar{\sigma}_m^{\mathcal{G}}\mu = (\mu(s_1), \ldots, \mu(s_{m-1}))\bar{\sigma}_m^{\mathcal{A}}$. Consequently,

$$(\mu(s_1), \ldots, \mu(s_{m-1}))\bar{\sigma}_m^{\mathcal{A}} = \{\mu(0), \mu(1)\}$$

is valid, for all $(s_1, \ldots, s_{m-1}) \in \{0,1\}^{m-1}$. By the definition of the α_0-product, the above equality implies

$$\{a_{0,k}, a_{1,k}\} \subseteq (a_{s_1,k}, \ldots, a_{s_{m-1},k})\sigma_{\mathbf{s},k}^{\mathcal{A}_k}$$

where $\mathbf{s} = (s_1, \ldots, s_{m-1})$ and

$$\sigma_{\mathbf{s},k} = \Psi_{mk}((a_{s_1,1}, \ldots, a_{s_1,k-1}), \ldots, (a_{s_{m-1},1}, \ldots, a_{s_{m-1},k-1}), \bar{\sigma}_m).$$

If $k = 1$, then $\sigma_{\mathbf{s},k} = \Psi_{m1}(\bar{\sigma}_m)$. If $k > 1$, then let us observe that, by the definition of k, $a_{s_t,j} = a_{0,j}$, $t = 1, \ldots, m-1$, is valid, for all j, $j = 1, \ldots, k-1$. Therefore,

$$\sigma_{\mathbf{s},k} = \Psi_{mk}((a_{0,1},\dots,a_{0,k-1}),\dots,(a_{0,1},\dots,a_{0,k-1}),\bar{\sigma}_m).$$

In both cases, we obtain that $\sigma_{\mathbf{s},k}$ does not depend on $\mathbf{s}$, and thus, there exists a $\sigma_m \in \Sigma_m^{(k)}$ such that

$$\{a_{0,k},a_{1,k}\} \subseteq (a_{s_1,k},\dots,a_{s_{m-1},k})\sigma_m^{\mathcal{A}_k}$$

holds, for all $(s_1,\dots,s_{m-1}) \in \{0,1\}^{m-1}$ which yields the validity of (a) if $m \neq 1$.

Now, let us suppose that $1 \in R$ and $m = 1$. By the definition of $\mathcal{G}$, there exists a $\bar{\sigma} \in \bar{\Sigma}_1$ such that $\bar{\sigma}^{\mathcal{G}}(0)$ and $\bar{\sigma}^{\mathcal{G}}(1)$ are valid. Since μ is an isomorphism, we obtain that $\bar{\sigma}^{\mathcal{A}}(\mu(0))$ and $\bar{\sigma}^{\mathcal{A}}(\mu(1))$ are also valid. Therefore, $\bar{\sigma}^{\mathcal{A}} = \{\mu(0),\mu(1)\}$. This equality implies $\{a_{0,k},a_{1,k}\} \subseteq \bar{\sigma}_1^{\mathcal{A}_k}$ where $\bar{\sigma}_1 = \Psi_{1k}(\bar{\sigma})$, and thus, $\mathcal{A}_k$ satisfies (a) in this case, too.

Regarding validity of (b), it follows from the proof of Theorem 1 in [13]. For the sake of completeness, we present its proof here as well. For this purpose, let us denote the set $\{k : 1 \leq k \leq n \;\&\; a_{0,k} \neq a_{1,k}\}$ by K. Obviously, $K \neq \emptyset$. Now, let $m \in R$ and $\mathbf{i} = (i_1,\dots,i_m) \in \{0,1\}^m$ be arbitrary elements. We distinguish the following two cases depending on m.

Case 1: $m > 1$. By the definition of $\mathcal{G}$, there is a $\bar{\sigma}_m \in \bar{\Sigma}_m$ with $(i_1,\dots,i_{m-1})\bar{\sigma}_m^{\mathcal{G}} = i_m$. Since μ is an isomorphism, this yields

$$(\mu(i_1),\dots,\mu(i_{m-1}))\bar{\sigma}_m^{\mathcal{A}} = \mu(i_m).$$

Therefore, $a_{i_m,k} \in (a_{i_1,k},\dots,a_{i_{m-1},k})\sigma_k^{\mathcal{A}_k}$ holds, for all $k \in K$, where

$$\sigma_k = \Psi_{mk}((a_{i_1,1},\dots,a_{i_1,k-1}),\dots,(a_{i_{m-1},1},\dots,a_{i_{m-1},k-1}),\bar{\sigma}_m).$$

But then there exists at least one index $l \in K$ such that

$$(a_{i_1,l},\dots,a_{i_{m-1},l})\sigma_l^{\mathcal{A}_l} \cap \{a_{0,l},a_{1,l}\} = \{a_{i_m,l}\}.$$

Consequently, $\mathcal{A}^{(l)}$ satisfies (b1) and (b2). To prove (b3), let $1 \neq u \in R$ and $\mathbf{s} = (s_1,\dots,s_{u-1}) \in \{0,1\}^{u-1}$ be arbitrary elements. By the definition of $\mathcal{G}$, there exists a $\sigma_{\mathbf{s}} \in \bar{\Sigma}_u$ with $(s_1,\dots,s_{u-1})\sigma_{\mathbf{s}}^{\mathcal{G}} = \{0,1\}$. Since μ is an isomorphism, this implies

$$(\mu(s_1),\dots,\mu(s_{u-1}))\sigma_{\mathbf{s}}^{\mathcal{A}} = \{\mu(0),\mu(1)\}.$$

Then $\{a_{0,k},a_{1,k}\} \subseteq (a_{s_1,k},\dots,a_{s_{u-1},k})\sigma_{\mathbf{s},k}^{\mathcal{A}_k}$ holds, for all $k \in K$, where

$$\sigma_{\mathbf{s},k} = \Psi_{uk}((a_{s_1,1},\dots,a_{s_1,k-1}),\dots,(a_{s_{u-1},1},\dots,a_{s_{u-1},k-1}),\sigma_{\mathbf{s}}).$$

Therefore, $\{a_{0,l},a_{1,l}\} \subseteq (a_{s_1,l},\dots,a_{s_{u-1},l})\sigma_{\mathbf{s},l}^{\mathcal{A}_l}$. If $1 \in R$ and $u = 1$, then, by the definition of $\mathcal{G}$, there is a $\sigma_1 \in \bar{\Sigma}_1$ with $\sigma_1^{\mathcal{G}} = \{0,1\}$. But then $\sigma_1^{\mathcal{A}} = \{\mu(0),\mu(1)\}$, and consequently, $\{a_{0,k},a_{1,k}\} \subseteq \bar{\sigma}_k^{\mathcal{A}_k}$, for all $k \in K$, where $\bar{\sigma}_k = \Psi_{1k}(\sigma_1)$. Thus

$\{a_{0,l}, a_{1,l}\} \subseteq \bar{\sigma}_l^{\mathcal{A}_l}$, i.e., $\mathcal{A}^{(l)}$ satisfies (b3) as well. This completes the proof of the necessity when $m \neq 1$.

Case 2: $1 \in R$ and $m = 1$. By the definition of $\mathcal{G}$, there is a $\bar{\sigma}_1 \in \bar{\Sigma}_1$ with $\bar{\sigma}_1^{\mathcal{G}} = i_1$. But then $\bar{\sigma}_1^{\mathcal{A}} = \mu(i_1)$. Therefore, $a_{i_1,k} \in \sigma_k^{\mathcal{A}_k}$ is valid, for all $k \in K$, where $\sigma_k = \Psi_{1k}(\bar{\sigma}_1)$. From this it follows that there exists at least one $l \in K$ such that

$$\sigma_l^{\mathcal{A}_l} \cap \{a_{0,l}, a_{1,l}\} = \{a_{i_1,l}\}.$$

Now, let $u \in R$ and $\mathring{\mathbf{s}} = (s_1, \ldots, s_{u-1}) \in \{0,1\}^{u-1}$ be fixed arbitrarily. In a similar way as above, it is easy to see that there is a $\sigma_{\mathbf{S},l} \in \Sigma_u^{(l)}$ such that $\{a_{0l}, a_{1l}\} \subseteq (a_{s_1 l}, \ldots, a_{s_{u-1} l})\sigma_{\mathbf{S},l}^{\mathcal{A}_l}$ if $u \neq 1$, and there is a $\sigma_l^* \in \Sigma_1^{(l)}$ with $\{a_{0,l}, a_{1,l}\}\sigma_l^{*\mathcal{A}_l}$ if $u = 1$. This completes the proof of the necessity.

For proving the sufficiency, let us assume that $\mathbf{B}$ satisfies the conditions of Theorem 1. Let us define the sets W and W' by

$$W = \{\{0,1\}^m : m \in R\} \quad \text{and} \quad W' = \{(i_1, \ldots, i_m) : (i_1, \ldots, i_m) \in W \ \&\ i_m = 0\}.$$

Let $|W'| = n$, and let γ denote a one-to-one mapping of $\{1, \ldots, n\}$ onto W'. By our assumption on $\mathbf{B}$, for any $p \in \{1, \ldots, n\}$, there exists a nondeterministic tree automaton $\mathcal{A}^{(\gamma(p))} = (A^{(\gamma(p))}, \Sigma^{(\gamma(p))}) \in \mathbf{B}$ satisfying conditions (b1), (b2), and (b3) with $\mathbf{i} = (i_1, \ldots, i_m) = \gamma(p)$ where $i_m = 0$. For the sake of simplicity, let us denote the elements $a_0^{(\gamma(p))}$ and $a_1^{(\gamma(p))}$ by 0 and 1, respectively. Furthermore, let us denote by $\mathcal{A}^* = (A^*, \Sigma^*)$ an automaton of $\mathbf{B}$ satisfying (a), moreover, let 0 and 1 denote the elements a_0^* and a_1^*, respectively.

Now, let $\mathcal{C} = (C, \Sigma) \in \mathbf{U}_R$ be an arbitrary nondeterministic tree automaton with $C = \{c_1, \ldots, c_r\}$. We prove that $\mathcal{C}$ can be embedded isomorphically into an α_0-product of nondeterministic tree automata from $\{\mathcal{A}^*\} \cup \{\mathcal{A}^{(\gamma(p))} : 1 \leq p \leq n\}$.

For this purpose, let us take all the r-dimensional column vectors over $\{0,1\}$ and order them in lexicographically increasing order. Let $\mathbf{Q}^{(r)}$ denote the matrix formed by these column vectors. Then $\mathbf{Q}^{(r)}$ is a matrix of type $r \times 2^r$ over $\{0,1\}$, the row vectors of $\mathbf{Q}^{(r)}$ are pairwise different, moreover, for any subset V of $\{1, \ldots, r\}$, there exists exactly one index $k \in \{1, \ldots, 2^r\}$ such that, for all $t \in \{1, \ldots, r\}$, $t \in V$ if and only if $q_{tk}^{(r)} = 0$. Let

$$\mathbf{Q} = (\mathbf{Q}^{(r)} \ldots \mathbf{Q}^{(r)})$$

where the number of the occurences of $\mathbf{Q}^{(r)}$ is $n+1$ in the partitioned form of $\mathbf{Q}$. Finally, let us define the one-to-one mapping μ of $\{c_1, \ldots, c_r\}$ onto the set of the row vectors of $\mathbf{Q}$ by $\mu(c_i) = (q_{i,1}, \ldots, q_{i,(n+1)2^r})$, $i = 1, \ldots, r$, and let $M = \{\mu(c_i) : i = 1, \ldots, r\}$.

Now, let us construct the α_0-product $\mathcal{A} = (A, \Sigma) =$

$$\underbrace{\mathcal{A}^* \times \cdots \times \mathcal{A}^*}_{2^r times} \times \underbrace{\mathcal{A}^{(\gamma(1))} \times \cdots \times \mathcal{A}^{(\gamma(1))}}_{2^r times} \times \cdots \times \underbrace{\mathcal{A}^{(\gamma(n))} \times \cdots \times \mathcal{A}^{(\gamma(n))}}_{2^r times}(\Sigma, \Psi)$$

in the following way. First of all, let us observe that $M \subseteq A$. To define the feedback functions, let us consider the following two cases.

Case 1: $1 \in R$ and $m = 1$. Let $\sigma \in \Sigma_1 (\subseteq \Sigma)$ be an arbitrary relational symbol, furthermore, let $\sigma^{\mathcal{C}} = \{c_{k_1}, \ldots, c_{k_l}\}$ where $0 \leq l \leq r$. Since $1 \in R$, the vector $\mathbf{i} = (0)$ is contained in W', and thus, there exists a $p_0 \in \{1, \ldots, n\}$ such that $\gamma(p_0) = (0)$. On the other hand, by the definition of $\mathbf{Q}^{(r)}$, there exists exactly one index $d \in \{1, \ldots, 2^r\}$ such that, for each $s \in \{0, \ldots, n\}$, the following assertion is valid:

$$\text{for all } t \in \{1, \ldots, r\},\ q_{t,s2^r+d} = 0 \text{ if and only if } t \in \{k_1, \ldots, k_l\}.$$

Now, the feedback functions Ψ_{1j}, $j = 1, \ldots, (n+1)2^r$, are defined as follows:

$$\Psi_{1j}(\sigma) = \begin{cases} \sigma_1 & \text{if } 1 \leq j \leq 2^r, \\ \sigma_{(0)} & \text{if } j = p_0 2^r + d, \\ \bar{\sigma}_{(0)} & \text{if } p_0 2^r < j \leq (p_0+1)2^r \ \ \& \ \ j \neq p_0 2^r + d, \\ \bar{\sigma}_{\gamma(p)} & \text{if } p_0 \neq p \in \{1, \ldots, n\} \ \ \& \ \ p2^r < j \leq (p+1)2^r, \end{cases}$$

where $\sigma_1 \in \Sigma_1^*$ satisfying (a), $\sigma_{(0)} \in \Sigma_1^{((0))}$ satisfying (b2), $\bar{\sigma}_{(0)} \in \Sigma_1^{((0))}$ satisfying (b3), finally, $\bar{\sigma}_{\gamma(p)} \in \Sigma_1^{(\gamma(p))}$ satisfying (b3).

Case 2: $1 \neq m \in R$. Let $\sigma \in \Sigma_m (\subseteq \Sigma)$ be an arbitrary m-ary relational symbol and let us consider $m-1$ elments from M denoted by $(q_{i_t,1}, \ldots, q_{i_t,(n+1)2^r})$, $t = 1, \ldots, m-1$. Then, $\mu(c_{i_t}) = (q_{i_t,1}, \ldots, q_{i_t,(n+1)2^r})$, $t = 1, \ldots, m-1$. Let us suppose that $(c_{i_1}, \ldots, c_{i_{m-1}})\sigma^{\mathcal{C}} = \{c_{k_1}, \ldots, c_{k_l}\}$ where $0 \leq l \leq r$. Then there is one and only one integer $d \in \{1, \ldots, 2^r\}$ such that, for every $s \in \{0, \ldots, n\}$, we have the following assertion:

$$\text{for all } t \in \{1, \ldots, r\},\ q_{t,s2^r+d} = 0 \text{ if and only if } t \in \{k_1, \ldots, k_l\}.$$

On the other hand, let us observe that, for any $v \in \{1, \ldots, 2^r\}$, the column vectors of $\mathbf{Q}$ with indices $s2^r+v$, $s = 0, \ldots, n$, are identical copies of some r-dimensional vector over $\{0,1\}$. Consequently, the vectors $(q_{i_1,s2^r+v}, \cdots, q_{i_{m-1},s2^r+v})$, $s = 0, \ldots, n$, are the copies of an $(m-1)$-dimensional vector over $\{0,1\}$. Let us denote the vector $(q_{i_1,v}, \ldots, q_{i_{m-1},v})$ by $\mathbf{s}_v$ if $1 \leq v \leq 2^r$, $v \neq d$, and the vector $(q_{i_1,d}, \ldots, q_{i_{m-1},d})$ by $(i'_1, \ldots, i'_{m-1})$. Let $\mathbf{i} = (i'_1, \ldots, i'_{m-1}, 0)$. Then $\mathbf{i} \in W'$, and thus, there is a $p_0 \in \{1, \ldots, n\}$ with $\gamma(p_0) = \mathbf{i}$. Now, we define the feedback functions as follows. For any $j \in \{1, \ldots, (n+1)2^r\}$, let

$$\Psi_{mj}((q_{i_1,1}, \ldots, q_{i_1,j-1}), \ldots, (q_{i_{m-1},1}, \ldots, q_{i_{m-1},j-1}), \sigma) =$$

$$= \begin{cases} \sigma_m & \text{if } 1 \leq j \leq 2^r, \\ \sigma_{\mathbf{i}} & \text{if } j = p_0 2^r + d, \\ \sigma_{\gamma(p),\mathbf{s}_v} & \text{if } j \neq p_0 2^r + d \;\&\; v \equiv j (\mathrm{mod}\ 2^r) \;\&\; p2^r < j \leq (p+1)2^r \\ & \quad \&\; p \in \{1, \ldots, n\} \end{cases}$$

where $\sigma_m \in \Sigma_m^*$ satisfying (a), $\sigma_{\mathbf{i}} \in \Sigma_m^{(\mathbf{i})}$ satisfying (b2), and $\sigma_{\gamma(p),\mathbf{s}_v} \in \Sigma_m^{(\gamma(p))}$ satisfying (b3).
In all the remaining cases, let us define the feedback functions Ψ_{mj} arbitrarily in accordance with the definition of the α_0-product.

Regarding above definition, we have to verify that it is really an α_0-product. If $1 \in R$ and $m = 1$, then our definition is obviously correct. Now, let $1 \neq m \in R$. Then Ψ_{mj} depends only on m if $1 \leq j \leq 2^r$. Let us consider the case when $2^r < j \leq (n+1)2^r$. Since the row vectors of $\mathbf{Q}^{(r)}$ are pairwise different, each element of M is uniquely determined by its first 2^r components. Therefore, the indices $i_1, \ldots, i_{m-1}$ are uniquely determined. Then $k_1, \ldots, k_l$ are determined by σ. Furthermore, d, $\mathbf{i}$ and p_0 are determined uniquely by $k_1, \ldots k_l$, the definition of $\mathbf{Q}^{(r)}$, and the first 2^r components of the elements in M under consideration. Now, if $j = p_0 2^r + d$, then the definition of Ψ_{mj} is in accordance with the definition of the α_0-product. If $j \neq p_0 2^r + d$, then j determines v and p uniquely, furthermore, $\mathbf{s}_v$ is determined by v and the first 2^r components of the considered elements of M. Consequently, the definition of Ψ_{mj} correspondes to the definition of the α_0-product in this case as well.

By the above observations, we have that $\mathcal{A}$ is an α_0-product of nondeterministic tree automata from $\{\mathcal{A}^*\} \cup \{\mathcal{A}^{(\gamma(p))} : 1 \leq p \leq n\}$. Let us consider the subautomaton of $\mathcal{A}$ determined by M and denote this subautomaton by $\mathcal{M} = (M, \Sigma)$. We prove that $\mathcal{C}$ and $\mathcal{M}$ are isomorphic, moreover, the mapping μ is a suitable isomorphism.

First, let us suppose that $1 \in R$ and $m = 1$. Let $\sigma \in \Sigma_1$ be an arbitrary relational symbol. We have to prove that $\sigma^{\mathcal{C}}(c_k)$ if and only if $\sigma^{\mathcal{M}}(\mu(c_k))$, for all $c_k \in C$, or equivalently, $\sigma^{\mathcal{C}}\mu = \sigma^{\mathcal{M}}$. We distinguish the following two cases.

Let us suppose that $\sigma^{\mathcal{C}} = \emptyset$. Then $d = 2^r$, furthermore, $\Psi_{1,(p_0+1)2^r}(\sigma) = \sigma_{(0)}$, and thus, the $(p_0+1)2^r$-th component of each element of $\sigma^{\mathcal{A}}$ is not equal to 1. On the other hand, the $(p_0+1)2^r$-th component of each element of M is equal to 1. Therefore, $\emptyset = \sigma^{\mathcal{A}} \cap M = \sigma^{\mathcal{M}}$. Conversely, let us assume that $\sigma^{\mathcal{M}} = \emptyset$. If $\sigma^{\mathcal{C}} \neq \emptyset$, then $\sigma^{\mathcal{C}} = \{c_{k_1}, \ldots c_{k_l}\}$ for some $1 \leq l \leq r$. Then, by the definition of Ψ_{1j}, $j = 1, \ldots, (n+1)2^r$, we obtain that

$$\sigma^{\mathcal{A}} \supseteq \{0,1\}^{p_0 2^r + d - 1} \times \{0\} \times \{0,1\}^{(n+1)2^r - p_0 2^r - d},$$

and the right-side set of the above inclusion contains $\mu(c_{k_t})$, for all t, $t = 1, \ldots, l$. Therefore, $\sigma^{\mathcal{A}} \cap M = \sigma^{\mathcal{M}} \neq \emptyset$ which is a contradiction. Consequently, $\sigma^{\mathcal{C}} = \emptyset$.

Now, let us suppose that $\sigma^{\mathcal{C}} = \{c_{k_1}, \ldots, c_{k_l}\}$ for some $1 \leq l \leq r$. Then

$$\sigma^{\mathcal{A}} \supseteq \{0,1\}^{p_0 2^r+d-1} \times \{0\} \times \{0,1\}^{(n+1)2^r-p_0 2^r-d},$$

and the right-side set contains $\mu(c_{k_t})$, for all t, $t = 1, \ldots, l$. On the other hand, by the definition of d, for all $t \in \{1, \ldots, r\}$, $q_{t,p_0 2^r+d} = 0$ if and only if $t \in \{k_1, \ldots, k_l\}$. This yields that $\sigma^{\mathcal{A}} \cap M = \{\mu(c_{k_1}), \ldots, \mu(c_{k_l})\}$, i.e., $\sigma^{\mathcal{M}} = \{\mu(c_{k_1}), \ldots, \mu(c_{k_l})\}$. Consequently, $\sigma^{\mathcal{C}}\mu = \sigma^{\mathcal{M}}$.

Now, let $1 \neq m \in R$, $\sigma \in \Sigma_m$, $c_{i_t} \in C$, $t = 1, \ldots, m-1$, be arbitrary elements. We have to show that

$$(c_{i_1}, \ldots, c_{i_{m-1}})\sigma^{\mathcal{C}}\mu = (\mu(c_{i_1}), \ldots, \mu(c_{i_{m-1}}))\sigma^{\mathcal{M}}$$

is valid. Let $(c_{i_1}, \ldots, c_{i_{m-1}})\sigma^{\mathcal{C}} = \{c_{k_1}, \ldots, c_{k_l}\}$ for some integer $0 \leq l \leq r$. Then, by the definition of Ψ_{mj}, $j = 1, \ldots, (n+1)2^r$,

$$(\mu(c_{i_1}), \ldots, \mu(c_{i_{m-1}}))\sigma^{\mathcal{A}} \supseteq \{0,1\}^{p_0 2^r+d-1} \times \{0\} \times \{0,1\}^{(n+1)2^r-p_0 2^r-d},$$

furthermore, $\{\mu(c_{k_1}), \ldots, \mu(c_{k_l})\} = \{(q_{k_t,1}, \ldots, q_{k_t,(n+1)2^r} : 1 \leq t \leq l\}$ is a subset of the right-side set. By the definition of d, for all $t \in \{1, \ldots, r\}$, $q_{t,p_0 2^r+d} = 0$ if and only if $t \in \{k_1, \ldots, k_l\}$. This yields that

$$(\mu(c_{i_1}), \ldots, \mu(c_{i_{m-1}}))\sigma^{\mathcal{A}} \cap M = \{(q_{k_t,1}, \ldots, q_{k_t,(n+1)2^r}) : 1 \leq t \leq l\} =$$

$$= \{\mu(c_{k_1}), \ldots, \mu(c_{k_l})\}.$$

Consequently, $(c_{i_1}, \ldots, c_{i_{m-1}})\sigma^{\mathcal{C}}\mu = (\mu(c_{i_1}), \ldots, \mu(c_{i_{m-1}}))\sigma^{\mathcal{M}}$, and thus, μ is an isomorphism of $\mathcal{C}$ onto $\mathcal{M}$.

This completes the proof of Theorem 1.

Remark. In particular, if $R = \{2\}$, then $\mathbf{U}_R$ is the class of the nondeterministic automata. Then as a special case of Theorem 1, we obtain the characterization of the isomorphically complete systems of nondeterministic automata with respect to the α_0-product which was presented in [14].

It is easy to observe that the nondeterministic tree automaton $\mathcal{G}$ satisfies the conditions of Theorem 1. Therefore, every nondeterministic tree automaton from $\mathbf{U}_R$ can be embedded into an α_0-power of $\mathcal{G}$. This implies the following corollary.

Corollary 1. *Every nondeterministic tree automaton from $\mathbf{U}_R$ can be embedded isomorphically into an α_0-product of suitable two-state nondeterministic tree automata.*

4 α_i-product with $i \geq 1$

In this section, we study the α_i-product with $i \geq 1$. For this reason, let $i > 0$ be an arbitrarily fixed integer. Then the isomorphically complete systems of nondeterministic tree automata with respect to the α_i-product can be characterized as follows.

Theorem 2. *A system* $\mathbf{B}$ *of nondeterministic tree automata from* $\mathbf{U}_R$ *is isomorphically complete for* $\mathbf{U}_R$ *with respect to the* α_i*-product if and only if, for all* $m \in R$ *and* $\mathbf{i} = (i_1, \ldots, i_m) \in \{0,1\}^m$, $\mathbf{B}$ *contains a nondeterministic tree automaton* $\mathcal{A}^{(\mathbf{i})} = (A^{(\mathbf{i})}, \Sigma^{(\mathbf{i})})$ *satisfying the following conditions:*

(I) $A^{(\mathbf{i})}$ *has two different elements* $a_0^{(\mathbf{i})}$ *and* $a_1^{(\mathbf{i})}$,

(II) *there exists a* $\sigma_{\mathbf{i}} \in \Sigma_m^{(\mathbf{i})}$ *with* $(a_{i_1}^{(\mathbf{i})}, \ldots, a_{i_{m-1}}^{(\mathbf{i})})\sigma_{\mathbf{i}}^{A^{(\mathbf{i})}} \cap \{a_0^{(\mathbf{i})}, a_1^{(\mathbf{i})}\} = \{a_{i_m}^{(\mathbf{i})}\}$,

(III) *for all* $1 \neq u \in R$ *and* $\mathbf{s} = (s_1, \ldots, s_{u-1}) \in \{0,1\}^{u-1}$, *there is a* $\sigma_{\mathbf{i},\mathbf{s}} \in \Sigma_u^{(\mathbf{i})}$ *for which* $\{a_0^{(\mathbf{i})}, a_1^{(\mathbf{i})}\} \subseteq (a_{s_1}^{(\mathbf{i})}, \ldots, a_{s_{u-1}}^{(\mathbf{i})})\sigma_{\mathbf{i},\mathbf{s}}^{A^{(\mathbf{i})}}$, *furthermore, there is a* $\bar{\sigma}_{\mathbf{i}} \in \Sigma_1^{(\mathbf{i})}$ *with* $\{a_0^{(\mathbf{i})}, a_1^{(\mathbf{i})}\} \subseteq \bar{\sigma}_{\mathbf{i}}^{A^{(\mathbf{i})}}$ *if* $1 \in R$.

Proof. The necessity of the conditions follows from Theorem 1 in [13]; the proof has the same idea as the proof of the necessity of (b) in Theorem 1 of Section 3. In order to prove the sufficiency, let us suppose that $\mathbf{B}$ satisfies the conditions of Theorem 2. Let us define the sets W and W' as above, i.e., let

$$W = \{\{0,1\}^m : m \in R\} \quad \text{and} \quad W' = \{(i_1, \ldots, i_m) : (i_1, \ldots, i_m) \in W \ \&\ i_m = 0\}.$$

Let $|W'| = n$, and let γ denote a one-to-one mapping of $\{1, \ldots, n\}$ onto W'. By our assumption on $\mathbf{B}$, for any $p \in \{1, \ldots, n\}$, there exists a nondeterministic tree automaton $\mathcal{A}^{(\gamma(p))} = (A^{(\gamma(p))}, \Sigma^{(\gamma(p))}) \in \mathbf{B}$ satisfying conditions (I), (II), and (III) with $\mathbf{i} = (i_1, \ldots, i_m) = \gamma(p)$ where $i_m = 0$. Again, let us denote the elements $a_0^{(\gamma(p))}$ and $a_1^{(\gamma(p))}$ by 0 and 1, respectively.

Now, let $\mathcal{C} = (C, \Sigma) \in \mathbf{U}_R$ be an arbitrary nondeterministic tree automaton with $C = \{c_1, \ldots, c_r\}$. We prove that $\mathcal{C}$ can be embedded isomorphically into an α_i-product of nondeterministic tree automata from $\{\mathcal{A}^{(\gamma(p))} : 1 \leq p \leq n\}$.

For this purpose, let

$$\mathbf{Q}' = (\mathbf{Q}^{(r)} \ldots \mathbf{Q}^{(r)})$$

where the number of the occurences of $\mathbf{Q}^{(r)}$ is $n+1$ in the partitioned form of $\mathbf{Q}'$. Furthermore, let us define the one-to-one mapping μ of $\{c_1, \ldots, c_r\}$ onto the set of the row vectors of $\mathbf{Q}'$ by $\mu(c_i) = (q_{i,1}, \ldots, q_{i,(n+1)2^r})$, $i = 1, \ldots, r$, and let $M' = \{\mu(c_i) : i = 1, \ldots, r\}$.

Let us construct the α_i-product $\mathcal{A} = (A, \Sigma) =$

$$\underbrace{\mathcal{A}^{(\gamma(1))} \times \cdots \times \mathcal{A}^{(\gamma(1))}}_{2^r times} \times \underbrace{\mathcal{A}^{(\gamma(1))} \times \cdots \times \mathcal{A}^{(\gamma(1))}}_{2^r times} \times \cdots \times \underbrace{\mathcal{A}^{(\gamma(n))} \times \cdots \times \mathcal{A}^{(\gamma(n))}}_{2^r times}(\Sigma, \Psi)$$

in the following way. First of all, let us observe that $M' \subseteq A$. To define the feedback functions, let us consider the following two cases.

Case 1: $1 \in R$ and $m = 1$. Let $\sigma \in \Sigma_1 (\subseteq \Sigma)$ be an arbitrary relational symbol, furthermore, let $\sigma^{\mathcal{C}} = \{c_{k_1}, \ldots, c_{k_l}\}$ where $0 \leq l \leq r$. Since $1 \in R$, the vector $\mathbf{i} = (0)$ is contained in W', and thus, there exists a $p_0 \in \{1, \ldots, n\}$ such that $\gamma(p_0) = (0)$. On the other hand, by the definition of $\mathbf{Q}^{(r)}$, there exists exactly one index $d \in \{1, \ldots, 2^r\}$ such that, for each $s \in \{0, \ldots, n\}$, the following assertion is valid:

$$\text{for all } t \in \{1, \ldots, r\},\ q_{t,s2^r+d} = 0 \text{ if and only if } t \in \{k_1, \ldots, k_l\}.$$

Let $j_0 = p_0 2^r + d$. Now, the feedback functions Ψ_{1j}, $j = 1, \ldots, (n+1)2^r$, are defined as follows:

$$\Psi_{1j}(\sigma) = \begin{cases} \bar{\sigma}_{\gamma(1)} & \text{if } 1 \leq j \leq 2^r, \\ \sigma_{(0)} & \text{if } j = j_0, \\ \bar{\sigma}_{\gamma(p)} & \text{if } j \neq j_0 \ \&\ p2^r < j \leq (p+1)2^r \text{ for some } p \in \{1, \ldots, n\}. \end{cases}$$

where $\bar{\sigma}_{\gamma(1)} \in \Sigma_1^{(\gamma(1))}$ satisfying (III), $\sigma_{(0)} \in \Sigma_1^{((0))}$ satisfying (II), and $\bar{\sigma}_{\gamma(p)} \in \Sigma_1^{(\gamma(p))}$ satisfying (III).

Case 2: $1 \neq m \in R$. Let $\sigma \in \Sigma_m (\subseteq \Sigma)$ be an arbitrary m-ary relational symbol and let us consider $m-1$ elments from M' denoted by $(q_{i_t,1}, \ldots, q_{i_t,(n+1)2^r})$, $t = 1, \ldots, m-1$. Then, $\mu(c_{i_t}) = (q_{i_t,1}, \ldots, q_{i_t,(n+1)2^r})$, $t = 1, \ldots, m-1$. Let us suppose that $(c_{i_1}, \ldots, c_{i_{m-1}})\sigma^{\mathcal{C}} = \{c_{k_1}, \ldots, c_{k_l}\}$ where $0 \leq l \leq r$. Then there is one and only one integer $d \in \{1, \ldots, 2^r\}$ such that, for every $s \in \{0, \ldots, n\}$, we have the following assertion:

$$\text{for all } t \in \{1, \ldots, r\},\ q_{t,s2^r+d} = 0 \text{ if and only if } t \in \{k_1, \ldots, k_l\}.$$

On the other hand, let us observe that, for any $v \in \{1, \ldots, 2^r\}$, the column vectors of $\mathbf{Q}'$ with indices $s2^r + v$, $s = 0, \ldots, n$, are identical copies of some r-dimensional vector over $\{0, 1\}$. Consequently, the vectors $(q_{i_1,s2^r+v}, \ldots, q_{i_{m-1},s2^r+v})$, $s = 0, \ldots, n$, are the copies of an $(m-1)$-dimensional vector over $\{0, 1\}$. Let us denote the vector $(q_{i_1,v}, \ldots, q_{i_{m-1},v})$ by $\mathbf{s}_v$ if $1 \leq v \leq 2^r$, $v \neq d$, and the vector $(q_{i_1,d}, \ldots, q_{i_{m-1},d})$ by $(i'_1, \ldots, i'_{m-1})$. Let $\mathbf{i} = (i'_1, \ldots, i'_{m-1}, 0)$. Then $\mathbf{i} \in W'$, and thus, there is a $p_0 \in \{1, \ldots, n\}$ with $\gamma(p_0) = \mathbf{i}$. Let $j_0 = p_0 2^r + d$ again. We define the feedback functions in the following way. For any $j \in \{1, \ldots, (n+1)2^r\}$, let

$$\Psi_{mj}((q_{i_1,1}, \ldots, q_{i_1,j+i-1}), \ldots, (q_{i_{m-1},1}, \ldots, q_{i_{m-1},j+i-1}), \sigma) =$$

$$= \begin{cases} \sigma_{\gamma(1),\mathbf{s}_j} & \text{if } 1 \le j \le 2^r, \\ \sigma_{\gamma(1),(i'_1,\ldots,i'_{m-1})} & \text{if } j = d, \\ \sigma_{\mathbf{i}} & \text{if } j = j_0, \\ \sigma_{\gamma(p),\mathbf{s}_v} & \text{if } j \ne j_0 \ \& \ v \equiv j \pmod{2^r} \ \& \ p2^r < j \le (p+1)2^r \\ & \text{for some; } p \in \{1,\ldots,n\}, \end{cases}$$

where $\sigma_{\gamma(1),\mathbf{s}_j}, \sigma_{\gamma(1),(i'_1,\ldots,i'_{m-1})} \in \Sigma_m^{(\gamma(1))}$ satisfying (III), $\sigma_{\mathbf{i}} \in \Sigma_m^{(\mathbf{i})}$ satisfying (II), and $\sigma_{\gamma(p),\mathbf{s}_v} \in \Sigma_m^{(\gamma(p))}$ satisfying (III). In all the remaining cases, let us define the feedback functions Ψ_{mj} in accordance with the definition of the α_i-product.

Regarding above definition, it is easy to verify that it is really an α_i-product, and thus, $\mathcal{A}$ is an α_i-product of nondeterministic tree automata from $\{\mathcal{A}^{(\gamma(p))} : 1 \le p \le n\}$. Let us consider the subautomaton of $\mathcal{A}$ determined by M'. Let $\mathcal{M}' = (M', \Sigma)$ denote this subautomaton. Then it is easy to prove that μ is an isomorphism of $\mathcal{C}$ onto $\mathcal{M}'$.

This completes the proof of Theorem 2.

Since the characterization of the isomorphically complete systems of nondeterministic tree automata with respect to the general product (see Theorem 1 in [13]) contains the same conditions as Theorem 2, we immediately obtain the following corollary.

Corollary 2. *The α_i-product is equivalent to the general product regarding isomorphically complete systems of nondeterministic tree automata provided that $i \ge 1$.*

References

[1] Z. Ésik, On isomorphic realization of automata with α_0-products, *Acta Cybernetica* **8** (1987), 119-127.

[2] F. Gécseg, Composition of automata, *Automata, Languages and Programming, 2nd Colloquium*, Saarbrücken, 1974, Lecture Notes in Computer Science (Springer-Verlag, Berlin Heidelberg New York Tokyo) **14** (1974), 351-363.

[3] F. Gécseg, *Products of Automata*, Springer-Verlag, Berlin Heidelberg New York Tokyo (1986).

[4] F. Gécseg and B. Imreh, On α_i-product of tree automata, *Acta Cybernetica* **8** (1987), 135-141.

[5] F. Gécseg and B. Imreh, On completeness of nondeterministic automata, *Acta Math. Hungar.* **68** (1995), 151-159.

[6] F. Gécseg and B. Imreh, On the cube-product of nondeterministic automata *Acta Sci. Math. (Szeged)* **60** (1995), 321-327.

[7] V. M. Glushkov, Abstract theory of automata, *Uspekhi Mat. Nauk*, **16:5 101** (1961), 3-62 (in Russian).

[8] G. Grätzer, *Universal Algebra*, 2nd edn. Springer-Verlag (New York Berlin Heidelberg Tokyo, 1979)

[9] B. Imreh, On α_i-products of automata, *Acta Cybernetica* **3** (1978), 301-307.

[10] B. Imreh, On complete systems of automata, in: *Proc. of the 2nd International Colloquium on Words, Languages and Combinatorics*, Kyoto, 1992, World Scientific (Singapore-New Jersey-London-Hong Kong), 1994, 207-215.

[11] B. Imreh, On a special composition of tree automata, *Acta Cybernetica* **10** (1992), 237-242.

[12] B. Imreh, Compositions of nondeterministic automata, RIMS Symposium on Semigroups, Formal Languages and Computer Science, Kyoto, 1996, to appear in RIMS Kokyuroku.

[13] B. Imreh, On isomorphic representation of nondeterministic tree automata, *Acta Cybernetica* **12** (1995), 11-21.

[14] B. Imreh and M. Ito, On α_i-product of nondeterministic automata, submitted to *Algebra Colloq.*

[15] M. Steinby, On the structure and realizations of tree automata, in *Second Coll. surles Arbres an Algèbre et en Programmation* Lille, 1979, 235-248.

Received July, 1996

Acta Cybernetica **13** (1997) 55-61.

On lexicographic enumeration of regular and context-free languages*

Erkki Mäkinen†

Abstract

We show that it is possible to efficiently enumerate the words of a regular language in lexicographic order. The time needed for generating the next word is $O(n)$ when enumerating words of length n. We also define a class of context-free languages for which efficient enumeration is possible.

1 Introduction

In [4] we considered the ranking and unranking algorithms for left Szilard languages of context-free grammars. These algorithms imply similar algorithms for context-free languages generated by arbitrary unambiguous context-free grammars. The present paper concerns a somewhat similar but more difficult problem of enumerating regular and context-free languages in lexicographic order. The widely studied problem of coding binary trees [3, 7] can be considered as a subproblem of our present problem. For example, in Zaks' coding method [7] we label the nodes and the leaves of a binary tree by 1 and 0, respectively. By traversing the tree in preorder we obtain a code word consisting of n (the number of nodes) 1's and $n+1$ 0's. The same set of words is obtained by considering the context-free language generated by productions $S \to 1SS$ and $S \to 0$. However, in the general case there are several nonterminals in the grammar in question. This means that the nodes in the corresponding derivation trees have different labels, and the problem of enumerating the "feasible codewords", i.e. the words in the language generated, is much more difficult.

2 Preliminaries

If not otherwise stated we follow the notations and definitions of [1]. Context-free grammars are denoted by $G = (V, \Sigma, P, S)$, where Σ is the set of terminals and V is the union of Σ and the set N of nonterminals.

*This work was supported by the Academy of Finland

†Department of Computer Science, University of Tampere, P.O. Box 607, FIN-33101 Tampere, Finland

If A is a nonterminal in a context-free grammar $G = (V, \Sigma, P, S)$, then $L(G, A)$ stands for the language derivable from A according to the productions of G. The length of a string β is denoted by $len(\beta)$.

For the sake of notational simplicity, we assume that context-free grammars are in Chomsky normal form (CNF), so that all productions are of the form $A \to BC$ or $A \to a$, where A, B, and C are nonterminals, and a is a terminal. The productions having A in their left hand side are called *A-productions*. We say that a production of the form $A \to a$ is *terminating*; the other productions are *continuing*. In a *regular grammar* [1] continuing productions have the form $A \to aB$.

When considering a lexicographic order in $L(G)$ generated by a context-free grammar $G = (V, \Sigma, P, S)$, we suppose that there is a total order $\prec_G$ defined in Σ which imposes the lexicographic order of the words in $L(G)$.

Throughout the paper, we use the unit-cost model for time and space. Hence, we suppose that normal arithmetic operations for arbitrary integers are possible in constant time and an arbitrary integer can be stored in one memory cell. All time and space bounds are given as functions of the length of words. The numbers of productions and nonterminals are always considered as constants.

3 Finding minimal words of given length

We first consider the problem of finding the lexicographically minimal words of different length in a given language. This problem is somewhat related to a very recently solved problem concerning the closure of context-free languages under min-operation. Namely, given a context-free language L, the language L_{min} is obtained by taking from all words of L of the same length only the first in lexicographic order [5]. Raz [6] has recently shown that L_{min} is context-free for an arbitrary context-free langauge L. Given a context-free grammar $G = (V, \Sigma, P, S)$, a total order $\prec_G$ in Σ, and a natural number n, our task in this section is to determine w such that $len(w) = n$ and $w \in L_{min}$.

In order to efficiently perform this task, we store in $A_{min}[i]$, for each nonterminal A and for each length $i = 1, \ldots, n-1$, the lexicographically minimal terminal string of length n obtainable from A according to the productions of G. Hence, each table entry $A_{min}[i]$ belongs to $L(G, A)_{min}$.

The following algorithm tabulates the A_{min} values for each nonterminal of the grammar in question. To simplify the notations, we suppose that Ω is not in Σ and we define $a \prec_G \Omega$ for all a in Σ. Ω will be used as a null value for undefined table entries. Moreover, we use the notation $conc(u, v)$ to stand for the normal concatenation of strings u and v, i.e. $conc(u, v) = uv$.

Algorithm 3.1 (Min)
Input: *A context-free grammar $G = (V, \Sigma, P, S)$, a total order $\prec_G$ in Σ, and a positive integer n.*
Output: *Table $A_{min}[1..n]$, for each nonterminal $A \in V \setminus \Sigma$; $min = S_{min}[n]$ is the minimal word of length n.*

Method:

for *each nonterminal* A **do**
  if *there is no terminating* A*-productions*
    then $A_{min}[1] \leftarrow \Omega$
    else $A_{min}[1] \leftarrow a$ *where* $a \prec_G b$ *holds for all other terminals* b *appearing in the right hand sides of terminating* A*-production;*
  for $i \leftarrow 2 \ldots n$ **do**
    for *each nonterminal* A **do**
    $min \leftarrow \Omega$*;*
    for *each continuing* A*-production* $A \rightarrow BC$ **do**
      for $j \leftarrow 1 \ldots i-1$ **do**
        if $B_{min}[j] \neq \Omega$ **and** $C_{min}[i-j] \neq \Omega$
        then
          if $conc(B_{min}[j], C_{min}[i-j]) \prec_G min$
          then $min \leftarrow conc(B_{min}[j], C_{min}[i-j])$
        od
      od
      $A_{min}[i] \leftarrow min$*;*
    od

End of Algorithm

As already mentioned, we consider the size of a grammar (including the numbers of terminals, nonterminals and productions) as a constant. Noticing this assumption it is clear that algorithm Min runs in time $O(n^2)$.

We also consider the total order $\prec_G^{-1}$ defined by letting $a \prec_G^{-1} b$ if and only if $b \prec_G a$. The minimal word in lexicographic order in $L(G)$ according to $\prec_G^{-1}$ is the maximal one according to $\prec_G$. This word is denoted by max (cf. min in Algorithm 3.1).

Theorem 3.1 *Let* G *be a context-free grammar. The words* min *and* max *of length* n *can be found in time* $O(n^2)$ *and in space* $O(n)$.

Theorem 3.1 can be sharpened if the input grammar is regular. Also the form of the algorithm changes a bit. Next, we rewrite the whole algorithm for the regular case.

Algorithm 3.2 (Reg-Min)

Input: *A regular grammar* $G = (V, \Sigma, P, S)$, *a total order* $\prec_G$ *in* Σ, *and a positive integer* n.
Output: *Table* $A_{min}[1..n]$, *for each nonterminal* $A \in V \setminus \Sigma$; $min = S_{min}[n]$ *is the minimal word of length* n.
Method:

for *each nonterminal* A **do**
  if *there is no terminating* A*-productions*
    then $A_{min}[1] \leftarrow \Omega$
    else $A_{min}[1] \leftarrow a$ *where* $a \prec_G b$ *holds for all other terminals* b *appearing in the right hand sides of terminating* A*-production;*
  for $i \leftarrow 2 \ldots n$ **do**
    for *each nonterminal* A **do**
    $min \leftarrow \Omega$;
    for *each continuing* A*-production* $A \rightarrow aB$ **do**
      if $B_{min}[i-1] \neq \Omega$
      then
        if $conc(a, B_{min}[i-1]) \prec_G min$
        then $min \leftarrow conc(a, B_{min}[i-1])$
        od
      od
      $A_{min}[i] \leftarrow min$;
    od

End of Algorithm

In Algorithm Reg_Min only a constant number of operations is needed for determining each table entry. Hence, we have the following theorem.

Theorem 3.2 *Let* G *be a regular grammar. The words* min *and* max *of length* n *can be found in* $O(n)$ *time and space.*

4 Enumeration of regular languages

So far, we have been able to find the minimal and maximal words in $L(G)$ of given length in lexicographic order. The algorithm enumerating the words in $L(G)$ of given length can now be given as follows using the words min and max:

Algorithm 4.1 (Enumerate)

Input: *A context-free grammar* $G = (V, \Sigma, P, S)$, *a total order* $\prec_G$ *in* Σ, *and a positive integer* n.
Output: *The words on length* n *in* $L(G)$ *in lexicographic order.*
Method:

present_word $\leftarrow$ *min;*
while *present_word* $\neq$ *max* **do**
find the next word in lexicographic order **od**

End of Algorithm

Obviously, our problem is to specify the step "find the next word in lexicographic order". We first consider the problem in the case of regular languages.

Suppose G is a regular grammar and $a_1a_2 \ldots a_n$ is a word in $L(G)$. We know that there is a deterministic finite automaton accepting $L(G)$ [1]. In terms of grammars this means that there is a regular grammar H such that $L(H) = L(G)$ and, for each nonterminal A, the terminals appearing in the right hand sides of A-productions are all different. Hence, without loss of generality, we can suppose that G has this property. It follows that we can conclude the sequence of nonterminals $S = A_1, A_2, \ldots, A_n$ needed in deriving the word $a_1a_2 \ldots a_n$ from the start symbol S, and further, we can conclude the sequence of productions applied.

We start from the end of $a_1a_2 \ldots a_n$ and look for a position in which we can replace the symbol a_i with a symbol b such that $a_i \prec_G b$.

The last symbol a_n is the only one in $a_1a_2 \ldots a_n$ produced by a terminating production. We first check whether or not there is a symbol b such that $A_n \rightarrow b$ is another terminating production and $a \prec_G b$. Provided that b is the first (according to $\prec_G$) such symbol we have found out that $a_1a_2 \ldots a_{n-1}b$ is the successor of $a_1a_2 \ldots a_n$. Otherwise (such b does not exist), we have to proceed further to the left.

Suppose now that a_i, $1 \leq i \leq n-1$, is the first symbol that can be replaced. This means that we have a continuing production $A_i \rightarrow bB$ such that $a_i \prec_G b$ (and b is before other such terminals according to $\prec_G$). If now $B_{min}[n-i]$ is defined, we can write the successor of $a_1a_2 \ldots a_n$ as

$$conc(a_1a_2 \ldots a_{i-1}b, B_{min}[n-i]).$$

Hence, when a symbol is changed then all positions in its right get the lowest possible value. If the B_{min} value is undefined for all possible B's appearing in the right hand sides of A_i-productions, we again have to proceed to the left.

If $a_1a_2 \ldots a_n \neq max$ then at least one of the symbols in $a_1a_2 \ldots a_n$ must be changeable. Since the number of productions is considered to be a constant, linear time is sufficient for finding the successor of a given word $a_1a_2 \ldots a_n$. Hence, we have the following theorem.

Theorem 4.1 *Given a regular grammar G, there is an algorithm for enumerating the words in $L(G)$ in lexicographic order such that the time needed for generating the next word is $O(n)$.*

Notice that the time bound of Theorem 4.1 holds also for the first word of the enumeration, i.e. for the minimal word in lexicographic order. This follows from Theorem 3.2.

5 Enumeration of context-free languages

In the previous section we were able to show that regular languages have an efficient enumeration algorithm. Unfortunately, it seems that the same does not hold for context-free langauges.

For the sake of simplicity, we suppose that context-free languages considered in the rest of the paper are generated by unambiguous context-free grammars. Suppose now that we apply the same approach as we used for regular languages. Hence, a word $a_1 a_2 \ldots a_n$ in $L(G)$ is given, and we first find out the sequence of productions used in the leftmost derivation producing the word. A unique derivation is always found because we suppose that G is unambiguous.

Let a_i be the symbol to be replaced with a symbol b having the property $a_i \prec_G b$. We have a leftmost derivation

$$S \Rightarrow \ldots \Rightarrow a_1 \ldots a_{i-1}\alpha \Rightarrow a_1 \ldots a_{i-1}a_i\beta$$

where β is a string of nonterminals such that $1 \leq len(\beta) \leq n-i$. We should now be able to efficiently find the lexicographically minimal word of length $n-i$ derivable from β. As in Algorithm 3.1 we have to check all possible combinations of the A_{min} table entries, for each nonterminal instance A appearing in β. In the general case, there seems to be no efficient solution for this problem.

On the other hand, an inefficient method can be implemented even without the preprocessing phase described in section 3: simply enumerate all the words in Σ^* and delete those not in $L(G)$.

We end this section by defining a subclass of context-free grammars which allow efficient enumeration of words in lexicographic order.

We say that a context-free grammar is *strongly prefix-free* if $L(G, A)$ is prefix-free for each nonterminal A. More formally, G is stronly prefix-free if derivations $A \Rightarrow^+ u$ and $A \Rightarrow^+ v$, where u and v are terminal strings, always imply that both $u = vw$ and $v = uw$ are impossible for all non-empty strings w. The class grammars generating left Szilard languages of context-free grammars [2] is an example of strongly prefix-free grammars.

Moreover, we say that a context-free grammar G is *length complete* if the following condition is fulfilled for each nonterminal A:

- if $w \in L(G, A)$, $len(w) = n$, then, for each i, $i = 1 \ldots n-1$, $L(G, A)$ contains a word of length i.

If G is stronly prefix-free then it is sufficient to maintain the A_{min} table values in lexicographic order and to consider only the minimal values from each table. This follows from the fact that in strongly prefix-free grammars the set of A_{min} values is always prefix-free. A_{min} values can be easily maintained in lexicographic order by using radix sort. Moreover, if G is length complete, then there is no need for backtracking because of lacking words of certain length.

The preprocessing phase (filling in the A_{min} tables) is now (asymptotically) as simple as with regular languages. Similarly, the next word can always be found

(asymptotically) as efficient as in the case of regular languages. Hence, we have the following theorem.

Theorem 5.1 *Given a stronly prefix-free, length complete context-free grammar G, there is an algorithm for enumerating the words in $L(G)$ in lexicographic order such that the time needed for generating the next word is $O(n)$.*

References

[1] M.A. Harrison, *Introduction to Formal Language Theory* (Addison-Wesley, 1978).

[2] E. Mäkinen, On context-free derivations. *Acta Universitatis Tamperensis* **197** (1985).

[3] E. Mäkinen, A survey on binary tree codings. *Comput. J.* **34** (1991) 438–443.

[4] E. Mäkinen, Ranking and unranking left Szilard languages. Dept. of Computer Science, University of Tampere. Report **A-1997-2**, January 1997.

[5] G. Păun and A. Salomaa, Closure properties of slender languages. *Theoret. Comput. Sci.* **120** (1993), 293–301.

[6] D. Raz, Context-free languages are closed under *min* operation. Manuscript, submitted for publication.

[7] S. Zaks, Lexicographic generation of ordered trees. *Theor. Comput. Sci.* **10** (1980) 63–82.

Received March, 1997

Acta Cybernetica **13** (1997) 63-76.

Bounded Space On-Line Variable-Sized Bin Packing*

Rainer E. Burkard† Guochuan Zhang†

Abstract

In this paper we consider the k-bounded space on-line bin packing problem. Some efficient approximation algorithms are described and analyzed. Selecting either the smallest or the largest available bin size to start a new bin as items arrive turns out to yield a worst-case performance bound of 2. By packing large items into appropriate bins, an efficient approximation algorithm is derived from k-bounded space on-line bin packing algorithms and its worst-case performance bounds is 1.7 for $k \geq 3$.

Keywords: On-line, bin packing, approximation algorithm.

1. Introduction

In the one-dimensional classical bin packing problem, a list L of items, i.e. numbers a_i $(i = 1, \cdots, n)$ in the range $(0, 1]$, are to be packed into bins, each of which has a capacity 1, and the goal is to minimize the number of bins used. Since the problem of finding an optimal packing is NP-hard, research has focused on finding near-optimal approximation algorithms. The classical bin packing problem and many of its variations are of fundamental importance, reflected in the impressive amount of research reported [1].

A bin packing algorithm is *on-line* if it packs items a_i solely on the basis of the sizes of the items a_j, $1 \leq j \leq i$ (i.e. the preceding items) and without any information on subsequent items.

For a list L of items and an on-line algorithm A, let $s(A, L)$ and $s(OPT, L)$ denote the total size of bins used by algorithm A and an optimal off-line algorithm, respectively. Then the worst-case performance bound of A is defined as

$$S_A^\infty = \lim_{k \to \infty} \sup_L \{s(A, L)/s(OPT, L) | s(OPT, L) \geq k\}$$

In classical bin packing $s(A, L)$ is just the number of bins used by algorithm A and $s(OPT, L)$ is the number of bins used by an optimal algorithm.

*This work was supported by Spezialforschungsbereich F 003 "Optimierung und Kontrolle", Projektbereich Diskrete Optimierung.

†TU Graz, Institut für Mathematik B, Steyrergasse 30, A-8010 Graz, Austria

In this paper, we pay our attention to the following two restrictions of on-line bin packing (For a rather complete survey on the worst case behaviour of on-line bin packing algorithms, see Galambos and Woeginger [7]).

Bounded space algorithms

We say, a bin becomes *active* (open), when it gets its first item. Once it is declared closed, it can never be active again. A bin packing algorithm uses *k-bounded space* if for each item a_i, the choice for the bin to pack it is restricted to a set of k or fewer active (open) bins.

Lee and Lee [10] proved that for every bounded space on-line bin packing algorithm A, $S_A^\infty \geq h_\infty = \sum_{i=1}^{\infty} 1/t_i \approx 1.69103$, where

$$t_1 = 1,\ t_{i+1} = t_i(t_i - 1), \quad \text{for } i \geq 1.$$

Galambos and Woeginger [6] even proved that the bound h_∞ could not be beaten by repacking.

Essentially, the following six types of bounded space on-line bin packing approximation algorithms have been studied.

(i) The first fit first close algorithm NkF ($k \geq 2$) is a simple extension of the *Next Fit* algorithm (Johnson [8]). Csirik and Imreh [3] constructed lists of items for which NkF is a factor $17/10 + 3/(10k - 10)$ away from the optimum. Mao [11] proved that this indeed is the worst that can happen. Hence $S_{NkF}^\infty = 17/10 + 3/(10k - 10)$ holds.

(ii) Mao [12] showed for the best fit first close algorithm ABF_k ($k \geq 2$) with bounded space k the performance bound $S_{ABF_k}^\infty = 17/10 + 3/(10k)$.

(iii) The best fit best close algorithm BBF_k ($k \geq 2$) was introduced by Csirik and Johnson [4]. They showed in a very sophisticated proof that "best is better than first", since independently of the value of k, always $S_{BBF_k}^\infty = 17/10$ holds.

(iv) Zhang [15] showed that for the first fit best close algorithm AFB_k ($k \geq 2$) which was also introduced by Csirik and Johnson [4], $S_{AFB_k}^\infty = 17/10 + 3/(10k - 10)$ holds.

(v) The $HARMONIC$ algorithm $HARM_k$ by Lee and Lee [10]. They showed that as k tends to infinity, $S_{HARM_k}^\infty$ tends to the number h_∞.

(vi) The $SIMPLIFIED\ HARMONIC$ algorithm SH_k by Woeginger [14], works similar to the $HARMONIC$ algorithm but uses another (more complicated) partition of the interval $(0, 1]$. Moreover, $S_{SH_k}^\infty \leq S_{HARM_k}^\infty$ for each $k \geq 2$.

Variable-sized on-line bin packing

Only few results are known concerning the more general problem in which bins need not be of a single given size [2,5,9,13,16]. The variable-sized bin packing problem is a variant of the classical bin packing, in which bin capacities may vary. We are given a list L of items, and several different types $B^1, \ldots, B^l$ of bins with sizes $1 = s(B^1) > s(B^2) > \cdots > s(B^l) > 0$. There is an inexhaustible supply of bins of each size. The goal is to pack the given items into the bins so that the sum of the sizes of the bins used is minimum. Observe that for the case that all bins

are of size one, this is just the classical one dimensional bin packing problem. This model is considerably more realistic than that of the classical problem.

In the on-line version of variable-sized bin packing, we cannot preview and rearrange the items of L before packing is started, but must instead accept and immediately pack each item as it arrives.

Friesen and Langston [5] gave three approximation algorithms with worst-case performance bounds of 2, 3/2, and 4/3. Only the first of these algorithms is on-line. Essentially, it is a simple modification of *Next Fit* and also has the same worst-case performance bound 2 as *Next Fit*. An off-line fully polynomial time approximation scheme has been devised by Murgolo [13] using a linear programming formulation of the problem. Kinnersley and Langston [9] presented fast on-line algorithms FFf for the variable-sized bin packing. They devised a scheme based on a user specific factor $f \geq \frac{1}{2}$ and proved that their strategy guarantees a worst-case performance bound not exceeding $1.5 + f/2 \geq 1.75$. By choosing $f = 1/2$, FFH, the best among FFf algorithms, is obtained. Zhang [16] proved that the tight bound of FFH is 1.7, the same bound as the *First Fit* algorithm in the classical bin packing. Csirik [2] derived an algorithm with worst-case performance bound of < 1.7 from the *Harmonic Fit* algorithm. To our knowledge, Csirik's algorithm is still the best up to now, for a short discussion see Section 4.

In this paper, we consider algorithms for on-line variable-sized bin packing problem with the added constraint that the algorithms can assign items only to one of k bins at a time. Two simple algorithms with bounds 2 are presented in Section 2. Section 3 analyses an algorithm with worst-case performance bound of 1.7 (for $k \geq 3$), which derived from bounded space on-line bin packing.

2. Some Simple Algorithms

When we design an algorithm for k-bounded space on-line variable-sized bin packing, we must answer the following questions.

- How to select the bin size when a new bin is required?
- Which bin among the k active bins is chosen for packing a_i?
- Which bin among the k active bins is closed when a new bin has to be created for a_i ?

For k-bounded space on-line bin packing, Csirik and Johnson [4] presented two packing rules and two closing rules. They are listed as follows.

P-FF Pack the current item a_i into the lowest indexed active bin that has enough space for it. Otherwise, open a new bin and place a_i in it.

P-BF Pack the current item a_i into the fullest active bin that has enough space for it. Otherwise open a new bin and place a_i in it.

C-FF Close the lowest indexed active bin.

C-BF Close the fullest active bin (with ties broken in favor of the lowest indexed bin).

We use $c(B)$ to denote the sum of the items in B. Given a list of $L = (a_1, \ldots, a_n)$, let $B_1, \ldots, B_m$ denote the list of bins ordered in a k-bounded space

on-line variable-sized algorithm. Let ALk(always largest) denote the algorithm which always uses only bins of size 1, i.e. bins of largest size, packs items using the P-FF or the P-BF rule, and closes bins using the C-FF or the C-BF rule.

Theorem 2.1. *$s(ALk, L) < 2s(OPT, L) + 1$, $k \geq 1$, for any list L.*

Proof. For $1 \leq i \leq m-1$, due to our packing rule $c(B_i) + c(B_{i+1}) > 1$. So,

$$\begin{aligned} s(ALk, L) &= (m-1) + 1 < 2\sum_{i=1}^{m} c(B_i) + 1 \\ &= 2\sum_{i=1}^{n} a_i + 1 \leq 2s(OPT, L) + 1. \ \square \end{aligned}$$

Any list consisting of items of size $\frac{1}{2} + \varepsilon$ and bins of size 1 and $\frac{1}{2} + \varepsilon$ for some arbitrarily small $\varepsilon > 0$, demonstrates that the bound of 2 is asymptotically tight for ALk.

While the worst-case behavior of *Best Fit* is superior to that of *First Fit* for the k-bounded space on-line bin packing problem, this is not the case for ALk where always the largest possible bins in variable-sized bin packing algorithm are used.

Now we consider the algorithm ASk(always smallest) which uses smallest possible bins, packs items using the P-FF or the P-BF rule, and closes bins using the C-FF or the C-BF rule.

Theorem 2.2. *$s(ASk, L) < 2s(OPT, L) + 1$, $k \geq 1$, for any list L.*

Proof. For $1 \leq i \leq m-1$, $c(B_i) + c(B_{i+1}) > s(B_i)$. Therefore, we have

$$\begin{aligned} s(ASk, L) &= \sum_{i=1}^{m} s(B_i) < 2\sum_{i=1}^{m} c(B_i) - c(B_1) - c(B_m) + s(B_m) \\ &< 2\sum_{i=1}^{n} a_i + 1 \leq 2s(OPT, L) + 1. \ \square \end{aligned}$$

Any list consisting of items of size $\frac{1}{2}$ and bins of size 1 and $1 - \varepsilon$ for some arbitrarily small $\varepsilon > 0$, demonstrates that the bound of 2 is asymptotically tight for ASk.

3. Algorithms Derived from k-Bounded Space On-Line Bin Packing

We start from the open, the packing and the closing rule.

Suppose that a_i is a large item (with size greater than 1/2). If it can be contained in a bin with size less than 1, it is called a B-item, else it is called an L-item. The smallest bin which can contain a large item a_i is called an a_i-home-bin. Obviously, if a_i is an L-item, then the size of the a_i-home-bin is 1.

Open rule: Suppose that the current item to be packed is a_i. If a_i is a B-item, then open an a_i-home-bin and pack a_i into it. Otherwise, start a new bin of size 1 for a_i.

Packing rules:P-FF and P-BF.

Closing rules:

C-VF: Close one active bin with size less than 1 if such a bin exists, otherwise use C-FF.

C-VB: Close one active bin with size less than 1 if such a bin exists, otherwise use C-BF.

Since we only have one open rule, we always use it to start a new bin. For any combination of a packing rule with a closing rule, we have four algorithms. The combination of P-FF with C-VF yields VFF_k and the combination of P-BF with C-VB yields VBB_k. Let VBF_k denote the (P-BF, C-VF) combination and VFB_k denote the (P-FF, C-VB) combination. In the following, we only analyze VBB_k algorithm. For the others, some remarks are given in the next section.

Theorem 3.1. *For any list L, we have*

$S^{\infty}_{VBB_k} = 1.7$, for $k \geq 3$.

Obviously, if we only have one type of bins, VBB_k is just BBF_k. From [4], we have $S^{\infty}_{VBB_k} \geq 1.7$, for $k \geq 3$.

We prove that the lower bound is tight with the help of the weighting function defined as follows.

We can divide all items in list L into 5 parts.

$A_1 = \{a \in L \mid 0 < a \leq 1/6\}$, $A_2 = \{a \in L \mid 1/6 < a \leq 1/3\}$,

$A_3 = \{a \in L \mid 1/3 < a \leq 1/2\}$, $A_4 = \{a \in L \mid a \text{ is a B-item}\}$,

$A_5 = \{a \in L \mid a \text{ is an L-item}\}$.

An item is called an A_i-item if it belongs to A_i, $i = 1, 2, 3, 4, 5$. A_4-items and A_5-items are large items.

Let us define a weighting function as follows.

$$W(a) = \begin{cases} (6/5)a, & \text{if } a \in A_1, \\ (9/5)a - 1/10, & \text{if } a \in A_2, \\ (6/5)a + 1/10, & \text{if } a \in A_3, \\ \max\{1.7a,\ s(\hat{B})\}, & \text{if } a \in A_4, \\ (6/5)a + 4/10, & \text{if } a \in A_5, \end{cases}$$

where $\hat{B}$ is the a-home-bin. $W(B)$, the weight of the bin B, is defined to be the sum of the weight of all items in bin B, i.e., $W(B) = \sum_{a_i \in B} W(a_i)$. And $W(L)$, the weight of the list L, is defined to be the sum of the weight of all items in L, i.e., $W(L) = \sum_{i=1}^{n} W(a_i)$. We are going to show that

$$s(VBB_k, L) - 4/5 \leq W(L) \leq 1.7 s(OPT, L).$$

Lemma 3.1. *For any list L, we have*

$W(L) \leq 1.7s(OPT, L)$.

Proof. Similar as in [16].

Lemma 3.2. *For any list L, we have*

$W(L) \geq s(VBB_k, L) - 4/5$, *for* $k \geq 3$.

Proof. It is obvious that at any time the size of at most one current active bin is not greater than $\frac{1}{2}$, when we use VBB_k. Note that except the last k bins, all of the bins used in a VBB_k packing are declared closed one by one. These bins are more than 1/2 full when they declared closed. In our analysis we first investigate the closed bins, thereafter we turn our attention to the last k bins.

Claim 3.1. *If a bin contains one A_5-item or two A_3-items, then its weight is not less than 1. If B_i is used in a VBB_k packing and $s(B_i) < 1$, then $W(B_i) \geq s(B_i)$.*

Proof. It is trivial. □

We shall analyze the packing as one active bin is to be closed. This active bin is called the currently closed bin. The case that the currently closed bin is not the lowest indexed active bin (the first active bin) will be considered in Claim 3.2. Further cases are treated in Claims 3.3 and 3.4.

Claim 3.2. *If the currently closed bin B_i is not the first active bin for a VBB_k packing, then $W(B_i) \geq s(B_i)$.*

Proof. Without loss of generality, let the current active bins be $B_1, \ldots, B_i, \ldots, B_k$, $i \neq 1$. By Claim 3.1 and inspection, we can assume that $s(B_i) = 1$ and B_i contains two items at least, and no A_5-item, one A_3-item at most. From the algorithm, B_i is the fullest bin at this time. If $c(B_1) \geq 5/6$, it is easy to see that $c(B_i) \geq c(B_1)$ and $W(B_i) \geq 1$. In the following, we only consider the case $B_1 < 5/6$. If B_i contains one B-item, B_i must contain another item α which can not be placed into B_1, i.e., $\alpha > 1/6$. Therefore we have $W(B_i) > 1.7(1/2) + (6/5)(1/6) > 1$. If B_i contains no B-item, we can also assume that $c(B_1) > 2/3$, otherwise B_i will belong to the special cases in Claim 3.1.

We only need to consider the two bottommost items of B_i, α and β.

Case 1. $\alpha \in A_2$, $\beta \in A_3$,

$$\begin{aligned} W(B_i) &\geq \frac{6}{5}c(B_i) + \frac{3}{5}\alpha > \frac{6}{5}c(B_i) + \frac{3}{5}(1 - c(B_1)) \\ &\geq \frac{6}{5}c(B_1) + \frac{3}{5} - \frac{3}{5}c(B_1) > \frac{3}{5} \cdot \frac{2}{3} + \frac{3}{5} = 1. \end{aligned}$$

Case 2. $\alpha \in A_2$, $\beta \in A_2$,

$$W(B_i) > \frac{6}{5}c(B_i) + \frac{3}{5}(1 - c(B_1)) \cdot 2 - \frac{1}{10} \cdot 2 \geq 1. \quad \square$$

In the following, we assume that the currently closed bin is the first active bin as it is declared closed.

Claim 3.3. *For a VBB_k packing, the currently closed bin B_i is just the first active bin and the next active bin is B_{i+1}. If $s(B_i) = s(B_{i+1}) = 1$ and $c(B_i) \geq c(B_{i+1}) > \frac{1}{2}$, then we have, when B_{i+1} contains no B-item,*

$$(6/5)c(B_i) + W(B_{i+1}) \geq 1 + (6/5)c(B_{i+1}) \tag{1}$$

and when B_{i+1} contains one B-item,

$$(6/5)c(B_i) + W(B_{i+1}) > 2. \tag{2}$$

Proof. When B_{i+1} contains one B-item, B_{i+1} must also contain at least one small item, say β, which has been accepted before the B-item. This follows from the fact that only bins with size < 1 can accept B-items as first items, but $s(B_{i+1}) = 1$. Clearly, $c(B_i)+\beta > 1$, otherwise β should be placed in bin B_i. Moreover, $w(B_{i+1}) > \frac{6}{5}\beta + 1.7 \cdot \frac{1}{2}$. The last two inequalities imply

$$\frac{6}{5}c(B_i) + W(B_{i+1}) > 6/5 + 1.7(1/2) > 2.$$

When B_{i+1} contains no B-item, we can assume that $\frac{1}{2} < c(B_i) < \frac{5}{6}$ and B_{i+1} contains no A_5-item and one A_3-item at most. Otherwise (1) is clear. Thus,

Case 1. $\frac{5}{6} > c(B_i) \geq \frac{2}{3}$

In this case, every item in B_{i+1} must be greater than $\frac{1}{6}$.

If B_{i+1} contains one A_3-item,

$$\begin{aligned}
\frac{6}{5}c(B_i) + W(B_{i+1}) &> \frac{6}{5}c(B_i) + \frac{6}{5}c(B_{i+1}) + \frac{3}{5}(1 - c(B_i)) \\
&= \frac{3}{5}c(B_i) + \frac{6}{5}c(B_{i+1}) + \frac{3}{5} \\
&\geq \frac{6}{5}c(B_{i+1}) + \frac{3}{5} \cdot \frac{2}{3} + \frac{3}{5} \\
&= 1 + \frac{6}{5}c(B_{i+1}).
\end{aligned}$$

If B_{i+1} contains no A_3-item then it is easy to see that B_{i+1} contains two A_2-items at least. Therefore

$$\begin{aligned}
\frac{6}{5}c(B_i) + W(B_{i+1}) &> \frac{6}{5}c(B_i) + \frac{6}{5}c(B_{i+1}) + \frac{3}{5}(1 - c(B_i)) \cdot 2 - \frac{1}{10} \cdot 2 \\
&= \frac{6}{5}c(B_{i+1}) + \frac{6}{5} - \frac{1}{5} \\
&= 1 + \frac{6}{5}c(B_{i+1}).
\end{aligned}$$

Case 2. $\frac{2}{3} > c(B_i) > \frac{1}{2}$

In this case, every item in B_{i+1} is greater than $\frac{1}{3}$, i.e., belongs to A_3 or A_5. Therefore, B_{i+1} must contain one A_5-item or two A_3-items. From Claim 3.1, $W(B_{i+1}) \geq 1$ but this is the easy case mentioned above. So, Claim 3.3 holds. □

Claim 3.4. *For a VBB_k packing ($k \geq 3$), the currently closed bin B_i is just the first active bin and the next active bin is B_{i+1}. Assume that $s(B_i) = 1$ and $c(B_i) < \frac{5}{6}$. Assume, moreover, that the sum of items in B_{i+1} is currently $\leq \frac{1}{2}$, but when the VBB_k packing is finished, $c(B_{i+1}) > 1/2$. Suppose that B_j with $s(B_j) = 1$ is the active bin next to B_{i+1}, when B_{i+1} accepts a new item γ after B_i has been closed.*

(i) *If B_j is closed before B_{i+1}, then*

$$\frac{6}{5}c(B_i) + W(B_{i+1}) + W(B_j) \geq 2 + \frac{6}{5}c(B_{i+1}). \tag{3}$$

(ii) *If B_{i+1} is closed before B_j, then*

$$\frac{6}{5}c(B_i) + W(B_{i+1}) + W(B_j) \geq 2 + \frac{6}{5}c(B_j). \tag{4}$$

Proof. If B_{i+1} contains at least two items when B_i is closed, then it is easy to prove that $\frac{6}{5}c(B_i) + W(B_{i+1}) \geq 1 + \frac{6}{5}c(B_{i+1})$ and (3) or (4) hold. Hence we only have to consider the case that B_{i+1} contains just one item α, when B_i is closed. We suppose that the bottommost item of B_j is β. Then α is greater than $\frac{1}{6}$ (if not, α can be placed into B_i) and $\beta > 1 - \alpha \geq \frac{1}{2}$. Afterwards, an item γ is placed into B_{i+1} and $c(B_j) + \gamma > 1$.

(i) B_j is closed before B_{i+1}.

• If $\frac{1}{6} < \alpha \leq \frac{1}{3}$, then, $c(B_j) \geq \beta > 1 - \alpha \geq \frac{2}{3}$, $c(B_i) > \frac{2}{3}$.

$$\begin{aligned} & \frac{6}{5}c(B_i) + W(B_j) + W(B_{i+1}) \\ > \; & \frac{6}{5}(c(B_i) + c(B_j)) + \frac{2}{5}' + \frac{6}{5}c(B_{i+1}) \\ > \; & \frac{6}{5} \cdot \frac{4}{3} + \frac{2}{5} + \frac{6}{5}c(B_{i+1}) \\ = \; & 2 + \frac{6}{5}c(B_{i+1}). \end{aligned}$$

• If $\frac{1}{3} < \alpha \leq \frac{1}{2}$ and $\frac{2}{3} < c(B_j) < \frac{5}{6}$, then, $\gamma > \frac{1}{6}$ and $\gamma + c(B_j) > 1$. When $\frac{1}{6} < \gamma \leq \frac{1}{3}$,

$$\begin{aligned} & \frac{6}{5}c(B_i) + W(B_j) + W(B_{i+1}) \\ \geq \; & \frac{6}{5}c(B_i) + \frac{6}{5}c(B_j) + \frac{2}{5} + \frac{6}{5}c(B_{i+1}) + \frac{3}{5}\gamma \\ > \; & \frac{6}{5} \cdot \frac{1}{2} + \frac{3}{5} \cdot \frac{2}{3} + \frac{2}{5} + \frac{3}{5} + \frac{6}{5}c(B_{i+1}) \\ = \; & 2 + \frac{6}{5}c(B_{i+1}). \end{aligned}$$

When $\frac{1}{3} < \gamma$,

$$\begin{aligned} & \frac{6}{5}c(B_i) + W(B_j) + W(B_{i+1}) \\ \geq\ & \frac{6}{5}c(B_i) + \frac{6}{5}c(B_j) + \frac{2}{5} + \frac{6}{5}c(B_{i+1}) + \frac{2}{10} \\ >\ & \frac{6}{5}\cdot\frac{1}{2} + \frac{6}{5}\cdot\frac{2}{3} + \frac{2}{5} + \frac{1}{5} + \frac{6}{5}c(B_{i+1}) \\ =\ & 2 + \frac{6}{5}c(B_{i+1}). \end{aligned}$$

- If $\frac{1}{3} < \alpha \leq \frac{1}{2}$ and $\frac{1}{2} < c(B_j) \leq \frac{2}{3}$, then, $\gamma > \frac{1}{3}$, $c(B_{i+1}) \geq \alpha + \gamma > \frac{2}{3}$ and $c(B_j) \geq c(B_{i+1})$. But this is impossible.
- If $\frac{5}{6} \leq c(B_j)$, then

$$\begin{aligned} & \frac{6}{5}c(B_i) + W(B_j) + W(B_{i+1}) \\ \geq\ & \frac{6}{5}c(B_i) + \frac{6}{5}c(B_j) + \frac{2}{5} + \frac{6}{5}c(B_{i+1}) \\ >\ & \frac{6}{5}\cdot\frac{1}{2} + \frac{6}{5}\cdot\frac{5}{6} + \frac{2}{5} + \frac{6}{5}c(B_{i+1}) \\ =\ & 2 + \frac{6}{5}c(B_{i+1}). \end{aligned}$$

(ii) B_{i+1} is closed before B_j.

It is clear that $c(B_{i+1}) \geq c(B_j)$ when B_{i+1} is closed. If $c(B_{i+1}) < \frac{2}{3}$ at this time, then $\beta \leq c(B_j) < \frac{2}{3}$. This implies that $\alpha > \frac{1}{3}$ and $\gamma > \frac{1}{3}$, i.e., $c(B_{i+1}) > \frac{2}{3}$. This is a contradiction. Therefore, we have $c(B_{i+1}) \geq \frac{2}{3}$.

- If $\frac{1}{6} < \alpha \leq \frac{1}{3}$, then

$$\begin{aligned} & \frac{6}{5}c(B_i) + W(B_{i+1}) + W(B_j) \\ >\ & \frac{6}{5}c(B_i) + \frac{6}{5}c(B_{i+1}) + \frac{3}{5}\alpha - \frac{1}{10} + \frac{2}{5} + \frac{6}{5}c(B_j) \\ >\ & \frac{6}{5}\cdot\frac{2}{3} + \frac{3}{5} + \frac{3}{5}\cdot\frac{1}{2} - \frac{1}{10} + \frac{2}{5} + \frac{6}{5}c(B_j) \\ =\ & 2 + \frac{6}{5}c(B_j). \end{aligned}$$

- If $\frac{1}{3} < \alpha \leq \frac{1}{2}$ and $\frac{1}{6} < \gamma \leq \frac{1}{3}$, then $\frac{2}{3} < c(B_j)$.

$$\begin{aligned} & \frac{6}{5}c(B_i) + W(B_{i+1}) + W(B_j) \\ \geq\ & \frac{6}{5}c(B_i) + \frac{6}{5}c(B_j) + \frac{2}{5} + \frac{6}{5}c(B_{i+1}) + \frac{3}{5}\gamma \\ >\ & \frac{6}{5}\cdot\frac{1}{2} + \frac{6}{5}c(B_j) + \frac{2}{5} + \frac{3}{5}(1 - c(B_j)) + \frac{6}{5}c(B_j) \\ >\ & \frac{3}{5} + \frac{3}{5}\cdot\frac{2}{3} + \frac{3}{5} + \frac{2}{5} + \frac{6}{5}c(B_j) \\ =\ & 2 + \frac{6}{5}c(B_j). \end{aligned}$$

- If $\frac{1}{3} < \alpha \leq \frac{1}{2}$ and $\frac{1}{3} < \gamma$, then

$$\begin{aligned} & \frac{6}{5}c(B_i) + W(B_{i+1}) + W(B_j) \\ \geq\ & \frac{6}{5}c(B_i) + \frac{6}{5}c(B_j) + \frac{2}{5} + \frac{6}{5}c(B_{i+1}) + \frac{2}{10} \\ >\ & \frac{6}{5}\cdot\frac{1}{2} + \frac{6}{5}\cdot\frac{2}{3} + \frac{2}{5} + \frac{1}{5} + \frac{6}{5}c(B_j) \\ =\ & 2 + \frac{6}{5}c(B_j). \end{aligned}$$

- If $\frac{1}{3} < \alpha \leq \frac{1}{2}$ and $\gamma \leq \frac{1}{6}$, then $c(B_{i+1}) \geq c(B_j) > \frac{5}{6}$.

$$\begin{aligned} & \frac{6}{5}c(B_i) + W(B_{i+1}) + W(B_j) \\ >\ & \frac{6}{5}c(B_i) + \frac{6}{5}c(B_{i+1}) + \frac{2}{5} + \frac{6}{5}c(B_j) \\ >\ & \frac{6}{5}\cdot\frac{1}{2} + \frac{6}{5}\cdot\frac{5}{6} + \frac{2}{5} + \frac{6}{5}c(B_j) \\ =\ & 2 + \frac{6}{5}c(B_j). \end{aligned}$$

Thus Claim 3.4 holds. □

Note 3.1. *If the hypothesis in Claim 3.4 does not hold, i.e., when the VBB_k packing ends, B_{i+1} is not greater than $\frac{1}{2}$ yet, then B_{i+1} belongs to the last k bins.*

Now, we consider the last k bins. Without loss of generality, suppose $S_1 = \{B_1, \ldots, B_m\}$ is the set of those closed bins (of size one) which are mentioned in Claim 3.3 and Claim 3.4. And suppose that B_i is closed before B_{i+1}, $i = 1, \ldots, m-1$. We also denote by $S_2 = \{B_{m+1}, \ldots, B_{m+t}\}$, $t \leq k$ those bins among the last k bins, whose weights are less than their sizes when the packing ends.

Note 3.2. *If the size of the first one of the last k bins is less than 1, then all of the closed bins have their sizes less than 1, i.e., $S_1 = \emptyset$, by the closing rule of our algorithm.*

To see this, if there are some closed bins with size 1, let B_p denote the last closed bin of size 1. After B_p is closed, the first active bin may not be a bin with size less than 1 by our algorithm. It is a contradiction.

Claim 3.5. *Each bin B used in VBB_k packing but not in $S_1 \cup S_2$ has a weight $W(B)$ not less than $s(B)$.*

Proof. Follows immediately from Claim 3.1 and Claim 3.2.

Claim 3.6. *If the case in Note 3.1 happens, then*

$$\begin{cases} \sum_{i=1}^{m+1} W(B_i) \geq m + (6/5)c(B_{m+1}), & \text{if } B_{m+1} \text{ contains no B-item,} \\ \sum_{i=1}^{m+1} W(B_i) \geq m + 1, & \text{if } B_{m+1} \text{ contains one B-item.} \end{cases}$$

Proof. Claim 3.6 follows directly from Claim 3.3 and Claim 3.4. □

Similarly, we get

Claim 3.7. *If the case in Note 3.1 does not happen, then*

$$\sum_{i=1}^{m+1} W(B_i) \geq m - 1 + (6/5)c(B_m) + W(B_{m+1}) \quad \text{for a } VBB_k \text{ packing.}$$

In the following, we consider S_2.

Claim 3.8. *When B_{m+1} contains one B-item,*

$$W(B_{m+2}) + \cdots + W(B_{m+t}) \geq t - 1 - 4/5.$$

Proof. We will consider two cases.

Case 1. $c(B_{m+2}), \ldots, c(B_{m+t}) > 1/2$.

(i) If both B_{m+i+1} and B_{m+i+2} contain one B-item each, $1 \leq i \leq t-2$, then

$$W(B_{m+i+1}) + W(B_{m+i+2}) > 2.$$

(ii) If B_{m+i+1} contains no B-item and B_{m+j+1} contains a B-item β, $1 \leq i < j \leq t-2$ then

$$(6/5)c(B_{m+i+1}) + W(B_{m+j+1}) > 2.$$

(iii) If B_{m+i+1} and B_{m+j+1} contain no B-item, $1 \leq i < j \leq t-2$ then

$$(6/5)c(B_{m+i+1}) + W(B_{m+j+1}) \geq 1 + (6/5)c(B_{m+j+1}).$$

(i), (ii) and (iii) can be proved in the similar way as in the proof of Claim 3.3. Therefore,

$$W(B_{m+2}) + \cdots + W(B_{m+t}) > t - 3 + \frac{1}{2} \cdot \frac{17}{10} + \frac{1}{2} \cdot \frac{6}{5} = (t-1) - \frac{11}{20}.$$

Case 2. There exists a bin $c(B_{m+j}) \leq 1/2$, $2 \leq j \leq t$. In this case, all bins following B_{m+j} contain one L-item, i.e., have their weights greater than 1. Thus it is easy to show that

$$W(B_{m+2}) + \cdots + W(B_{m+t}) > (t-1) - 4/5. \ \square$$

Similarly, it can be shown

Claim 3.9. *When $c(B_{m+1}) > 1/2$ contains no B-item,*

$$(6/5)c(B_{m+1}) + W(B_{m+2}) + \cdots + W(B_{m+t}) \geq t - 4/5. \ \square$$

Claim 3.10. *If $c(B_{m+1}) \leq 1/2$, when the packing ends, then*

$$\sum_{i=m+1}^{m+t} W(B_i) + (6/5)c(B_m) > t + 1 - 4/5.$$

Proof. If $c(B_{m+1}) \leq 1/2$, when the packing ends, then B_j contains one A_5-item for $j = m+2, \ldots, m+t$. Therefore,

$$\sum_{i=m+1}^{m+t} W(B_i) + (6/5)c(B_m) > t - 1 + (6/5)(c(B_m) + c(B_{m+1})) > t - 4/5. \ \square$$

By Claims 3.5 to 3.10, we get $W(L) \geq s(VBB_k, L) - 4/5$. This completes the proof of Lemma 3.1. $\square$

Combining Lemma 3.1 with Lemma 3.2, we have $S^{\infty}_{VBB_k} \leq 1.7$, for $k \geq 3$. Therefore, we conclude that

$$S^{\infty}_{VBB_k} = 1.7, \ k \geq 3.$$

Thus Theorem 3.1 is proven. $\square$

4. Conclusions and Remarks

This paper deals with an on-line variable-sized bin packing problem which uses bounded space. Up to now, the best bound of the known on-line variable-sized bin packing algorithms [2] is a bit smaller than 1.7. In fact, the algorithm so-called VH_M in [2] which is derived from *Harmonic Fit* is a *bounded space* algorithm. Let $M > 1$ be a positive integer and let $M_j = \lceil M \cdot s(B^j) \rceil$ $(j = 1, 2, \ldots, l)$. Then the algorithm uses k-bounded space where

$$k = M_1 + M_2 + \cdots + M_l - l + 1.$$

When $M_l \leq 5$, the worst-case performance bound of VH_M is greater than 1.7. If and only if $M_l \geq 7$, VH_M can do better than VBB_k where $k \geq 6l+1$. To see VBB_k is efficient, we observe that it only needs $k \geq 3$ to reach the bound 1.7 which does not depend on the number l of bin sizes.

We can also analyse the other three algorithms VFB_k, VFF_k and VBF_k with the similar technique. For example, with a modified weighting function

$$W^*(a) = \begin{cases} \max\{1.7a,\ s(\hat{B})\} + 3/(10k-10), & \text{if} \quad a \in A_4, \\ (6/5)a + 4/10 + 3/(10k-10), & \text{if} \quad a \in A_5, \\ W(a), & \text{otherwise} \end{cases}$$

where $W(a)$ is defined in Section 3, we can prove that $S^{\infty}_{VFB_k} = 1.7 + \frac{3}{10(k-1)}$, for $k \geq 2$. It is clear that all the three algorithms can not beat algorithm VBB_k.

Acknowledgement

The authors wish to thank the anonymous referee for his helpful comments on an earlier version of this paper.

References

[1] E.G. Coffman, Jr, M.R. Garey and D.S. Johnson, Approximation Algorithms for Bin Packing: An Updated Survey. In Analysis and Design of Algorithms in Combinatorial Optimization, Ausiello and Lucertini (eds), Springer, New York, 49-106 (1984).

[2] J. Csirik, An On-Line Algorithm for Variable-Sized Bin Packing. *Acta Informatica* **26,** 697-709 (1989).

[3] J. Csirik and B. Imreh, On the Worst-Case Performance of the NkF Bin Packing Heuristic. *Acta Cybernetica* **9,** 89-105 (1989).

[4] J. Csirik and D.S. Johnson, Bounded Space On-Line Bin Packing: Best is Better than First. The Proceedings of 2nd Annual ACM-SIAM Symposium on Discrete Algorithms, 309-319 (1991).

[5] D.K. Friesen and M.A. Langston, Variable Sized Bin Packing. *SIAM J. Comput.* **15,** 222-229 (1986).

[6] G. Galambos and G.J. Woeginger, Repacking Helps in Bounded Space On-Line Bin Packing, *Computing* **49,** 329-338 (1993).

[7] G. Galambos and G.J. Woeginger, On-Line Bin Packing – A Restricted Survey. *Zeitschrift für Operations Research* **42,** 25-45 (1995).

[8] D.S. Johnson, Fast Algorithms for Bin Packing. *J. Comput. System Sci.* **8,** 272-314 (1974).

[9] N.G. Kinnersley and M.A. Langston, Online Variable-Sized Bin Packing. *Discrete Applied Mathematics* **22**, 143-148 (1988/89).

[10] C.C. Lee and D.T. Lee, A Simple On-Line Bin Packing Algorithm, *J. Assoc. Comput. Mach.* **32**, 562-572 (1985).

[11] W. Mao, Tight Worst-Case Performance Bounds for Next-k-Fit Bin Packing. *SIAM J. on Computing*, **22**, 46-56 (1993).

[12] W. Mao, Best-k-Fit Bin Packing. *Computing* **50**, 265-270 (1993).

[13] F.D. Murgolo, An Efficient Approximation Scheme for Variable-Sized Bin Packing. *SIAM. J. Comput.* **16**, 149-161 (1987).

[14] G.J. Woeginger, Improved Space for Bounded Space On-Line Bin Packing Algorithms, *SIAM J. Discr. Math.* **6**, 575-581 (1993).

[15] G.C. Zhang, Tight Worst-Case Performance Bound For AFB_k Bin Packing. Technical Report No. 015 in Inst. of Appl. Math., Academia Sinica (1994). To appear in *Acta Mathematicae Applicatae Sinica.*

[16] G.C. Zhang, Worst-Case Analysis of the FFH Algorithm for On-Line Variable-Sized Bin Packing. *Computing* **56**, 165-172 (1996).

Received March, 1996

Acta Cybernetica **13** (1997) 77-83.

Generalized Harary Games*

András Pluhár †

Abstract

There are a number of positional games known on the infinite chessboard. One of the most studied is the 5-*in-a-row*, whose rules are almost identical to the ancient Japanese *Go-Moku*. Along this line Harary asked if a player can achieve a translated copy of a given polymino P when the two players alternately take the squares of the board. Here we pose his question for general subsets of the board, and give a condition under which a draw is possible. Since a drawing strategy corresponds to a good 2-coloration of the underlying hypergraph, our result can be viewed as a derandomization of the Lovász Local Lemma.

1 Introduction and Results

Frank Harary proposed the following game on the infinite chessboard (two dimensional lattice) which resembles both the k-in-a-row, and the Hex (see [5], [8]). Let us recall that a *polymino* is a set of connected squares of the chessboard. Given a polymino P, the players, I and II take one square of the chessboard at each turn. I tries to take a translated copy of P, while II's goal is to prevent I from doing this. A polymino P is a *winner* if I has a winning strategy, otherwise P is a *loser*. Andreas Blass found most of the minimal loser polyminoes, using Hales-Jewett-type of pairings (see [5]). The opposite task, that is to decide about the winners, was carried out exhibiting some sequences of winning moves. Practically all of the winner polyminoes are known, there are at most twelve of them. The status of the largest one, called *snaky*, is still unsettled. Although it seems to be a winner, no one has found a convincing proof yet (see [5]).
In this paper we are interested in a more general situation:

1. II has to prevent I from taking not only one, but several other polyminoes,
2. P is not necessary a polymino (i.e. connected set of squares).

*The author was partially supported by the Austrian-Hungarian Action Fund "Combinatorial optimization - research and education" 20u2 and the Bilateral Intergovermental S & T Cooperation between Italy and Hungary I-14/95.

†Department of Computer Science, Attila József University, Árpád tér 2, H-6720 Szeged, Hungary

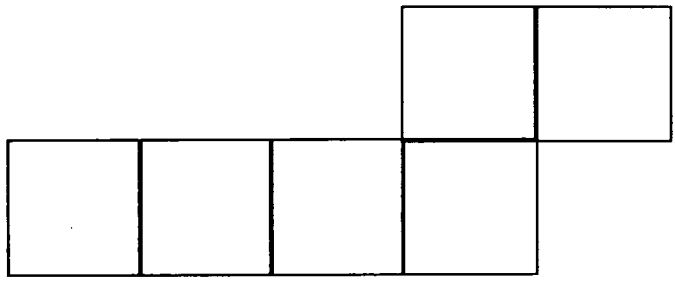


Figure 1: The polymino called snaky

The methods, used in the solution of the original problem, break down hopelessly in the generalized cases. (Especially, when both modifications are considered.) If we have some additional properties of the winning patterns, then the weight function technique still can provide an answer. First we need to formalize the notion of a *hypergraph game.*

Definitions:
An (X, H) pair is a *hypergraph* if $H \subset 2^X$. Given a hypergraph (X, H) the players I and II can play the following game that we call (p, q, H) *hypergraph game* (or shortly (p, q, H) *game*):
I and II select p and q unselected elements of X in each turn, respectively. The first, who selects all elements of an $A \in H$, wins.
For $1 \leq i \leq k$ let P_i be a set of squares of the infinite chessboard, $n_i = |P_i|$ the number of elements in P_i, and $d(P_i)$ the diameter of P_i in Euclidean norm. Let $A(p, q; \mathcal{P})$ be an (p, q, H) game, where X is the infinite board and H consists of the translated copies of $P_1, ..., P_k$. Furthermore, set $n = min\{n_i : 1 \leq i \leq k\}$ and $d = max\{d(P_i) : 1 \leq i \leq k\}$.

Theorem 1 *If* $n \geq 50 \log_2 d + 25 \log_2 k + 25$, *then* II *can prevent* I *from winning the game* $A(1, 1; \mathcal{P})$.

It is natural to ask what happens in general, that is if I wins the game $A(p, q; \mathcal{P})$, or it is a draw. A similar argument like in [10] would show that I wins every $A(p, q; \mathcal{P})$ if $p > q$. (Omitting the details, we give just a sketch of I's strategy: Take squares far from each other for some turn. In the subsequent turns neglect those which are "too close" to squares taken by II. Since $p > q$, I can build up an arbitrary pattern.) Hence the only open cases where $p \leq q$. The most intriguing case the $p = q = 1$, although we believe that dropping the diameter restriction does not really help I, which is spelled out in Conjecture 1.

Conjecture 1 *There exists a function* $f(k)$ *depending only on* k *such that, if* $n \geq f(k)$, *then* II *can prevent* I *from winning the game* $A(1, 1; \mathcal{P})$.

2 Weight Functions

In this section we shall recall the most useful method in the theory of hypergraph games, the method of *weight functions*. It is impossible to trace back when it did first appear, in some sense the idea is as old as the exponential function. It has surfaced in the recreational mathematics several times, eventually P. Erdős and J. L. Selfridge put it to its proper place in mathematics (see [7]). Later a large number of applications were found, both in the theory of games and in other parts of discrete mathematics (see for example [1, 2, 3, 4]). The following result may be called as the "Fundamental Theorem of Hypergraph Games"; for the case $p = q = 1$ it was proved in [7], the general form is from [2].

Theorem 2 *[2, 7] II can prevent I from winning the (p, q, H)-game if*

$$\sum_{A \in H} (1+q)^{-\frac{|A|}{p}} < \frac{1}{1+q}.$$

This theorem cannot be used in our case directly, since $\sum_{A \in H} 2^{-|A|}$ is *not* finite. Yet another difference is that it does not harm the player II if he gets extra elements from X. Intuitively it is clear that II is better off if in some turns he can take more than q elements of X (even if he receives the extra elements randomly). We call a hypergraph game *relaxed* (p, q, H) game if at each turn I takes at most p, while II takes at least q elements of X. Theorem 2 holds for the relaxed (p, q, H) games, too. For the sake of compactness, we repeat Beck's proof from [2], getting Lemma 1.

Lemma 1 *[2] II can prevent I from winning the relaxed $(p, 1, H)$-game if*

$$\sum_{A \in H} 2^{-\frac{|A|}{p}} < \frac{1}{2}.$$

Proof of Lemma 1.
For any $A \in H$ let $A_k(I)$ and $A_k(II)$ be the number of elements in A, after I's k^{th} move, selected by I and II, respectively. Furthermore

$$w_k(A) = \begin{cases} \lambda^{-|A|+A_k(I)} \text{ if } A_k(II) = 0 \\ 0 \text{ otherwise} \end{cases},$$

where $\lambda > 0$ and for any $x \in X$

$$w_k(x) = \sum_{x \in A,\ A \in H} w_k(A).$$

The numbers $w_k(A)$ and $w_k(x)$ are called the *weight* of A and x (in the k^{th} step), respectively.
For selecting an element in the k^{th} step II uses the *greedy* algorithm, i.e. he

chooses an unselected element $y^k \in X$ of maximum weight. Let $x_1^{k+1}, ..., x_p^{k+1}$ be the elements selected by I in the $(k+1)^{st}$ step and let

$$w_k = \sum_{A \in H} w_k(A)$$

be the *total sum* or *potential.* The following inequality holds for the potential:

$$w_k - w_k(y^k) + (\lambda^p - 1)w_k(y^k) \geq w_{k+1}$$

if $k \geq 0$. Indeed, w_k decreases by $w_k(y^k)$ upon selecting y^k. On the other hand, it is easy to see that the increase of the potential, caused by I's newly selected elements, is the greatest in the case where:

1. $w_k(x_l^{k+1})$ is maximal for $1 \leq l \leq p$
and
2. if $w_k(A) \neq 0$ $(A \in H)$, then $x_l^{k+1} \in A$ iff $x_m^{(k+1)} \in A$, $1 \leq l$ and $m \leq p$.

But the increase in this case is just $(\lambda^p - 1)w_k(y^k)$, therefore the inequality is proved. Setting $\lambda = 2^{1/p}$, we get

$$w_k \geq w_{k+1},$$

for $k \geq 0$, which justifies that w_k is called potential.
Particularly

$$w_1 \leq 2(\lambda^p) \sum_{A \in H} 2^{-|A|} < 1.$$

Let us suppose that I wins the game in the k^{th} step, occupying the set A. This would imply

$$w_k \geq \lambda^{-|A|+A_k(I)} = 0,$$

which contradicts the monotonicity of the potential. □

Remarks.
1. Intuitively, the potential measures the overall danger that the vertices of the elements of H are being selected by I during the game. Most often it is done by choosing an appropriate exponential function, and this exponentiality is to which one can attribute the power of the weight function method. Practically speaking, one may expect reasonable theorems via weight functions for a family of hypergraphs

$$\mathcal{F} = \{(X, H) : \gamma \in \Gamma\}$$

if there exists a polynomial p, such that

$$|H_\gamma| \leq p(|X_\gamma|)$$

for all $\gamma \in \Gamma$. As we shall see, the special structure of the hypergraphs can also help, even when $|H_\gamma| = \infty$.

2. There is a deep connection between the random 2-colorings of a hypergraph (X, H) and the $(1, 1, H)$-game. The inequality

$$\sum_{A \in H} 2^{-|A|} < \frac{1}{2}$$

says that the expected number of monochromatic elements of H is less than 1, i.e. there is a good 2-coloring. From this point of view the weight function technique is nothing else but the *derandomization* of the well known first moment method. The conditional probabilities for certain events can also be interpreted as weight function values (see [1]). This method makes it possible to turn probabilistic algorithms into effective, polynomial time deterministic ones (see [4]). On the other hand, if the expected number of monochromatic sets is "small", then II might achieve a draw in the corresponding hypergraph game.

3 Proof of Theorem 1

First we cut up the board into $d \times d$ squares, and call the set of these squares $\mathcal{S}$. For an $S \in \mathcal{S}$ let $\bar{S}$ be the union of S and the other eight $d \times d$ squares surrounding S. We shall refer to $\bar{S}$'s as the sub-boards. For an $\bar{S}$ the game $(\bar{S}, H(\bar{S}))$ is the hypergraph game, where the winning sets are those elements of H, which lie entirely in $\bar{S}$. If II plays a strategy which prevents I from winning any of the $(\bar{S}, H(\bar{S}))$ for $S \in \mathcal{S}$, then it prevents I from winning the $A(1, 1; \mathcal{P})$ also. Indeed, let us suppose that I succeeds in taking all elements (squares) of a translated copy of P_i for some $i = 1, ..., k$. If this copy of P_i has a common element with an $S \in \mathcal{S}$, then, from the diameter limitation, the whole copy lies within $\bar{S}$, i.e. I wins the game $(\bar{S}, H(\bar{S}))$, too.

One of the difficulties in establishing a strategy which guarantees a draw for II on every sub-board $\bar{S}$ is that the sub-boards are not disjoint. It means I's mark appears on nine of the sub-boards, and although II's answer is ninefold too, we cannot expect it to be the best on all of these sub-boards. We shall just ignore eight of them, and concentrating on one of them at a time, we create a relaxed $(25, 1; H(\bar{S}))$ game on every sub-board $\bar{S}$. Similarly to the idea of Lemma 3 of [10], we define a relation $\mathcal{O}$ on the set of the sub-boards, and use it to decide which sub-board should receive the mark of II. At the beginning of the game $\mathcal{O}$ is empty. We say $\bar{S}_u$ *owes* $\bar{S}_w$ if $(\bar{S}_u, \bar{S}_w) \in \mathcal{O}$. At the l^{th} step II selects a sub-board $\bar{S}^*$ such that:

1. $\bar{S}^*$ contains x_l, the last selection of I,
and
2. $\bar{S}^*$ does not owe any sub-board $\bar{S} \ni x_l$.

Then II updates the relation $\mathcal{O}$. $\bar{S}^*$ owes all $\bar{S} \neq \bar{S}^*$ which contain x_l, and non of these $\bar{S}$'s owe $\bar{S}^*$ in the updated relation. Now, if a sub-board $\bar{S} \ni x_l$ was not selected, then a (say) $\bar{S}^*$ was. $\bar{S}^*$ owes $\bar{S}$, and cannot be selected again until $\bar{S}$ is

selected. Since at most 24 sub-boards may owe a sub-board $\bar{S}$, at least every 25^{th} step on every sub-board is answered.
Within a sub-board $\bar{S}$, selected by the previous rule, II plays accordingly to Lemma 1. It gives that II draws in *all* relaxed $(\bar{S}, H(\bar{S}))$ game, provided that

$$\sum_{A \in H(\bar{S})} 2^{-\frac{|A|}{25}} < \frac{1}{2}.$$

On the other hand $|H(\bar{S})| < 9d^2k$, so if

$$n \geq 50 \log_2 d + 25 \log_2 k + 100,$$

then the above inequality holds, therefore I cannot win. □

4 Conclusion

Upon proving Theorem 1, we have reached the limits of the weight function technique. On one hand, there is no reason to believe that winning sets of larger and larger diameter would really benefit I. On the other hand, the weight functions, unless an ingenious idea is incorporated, cannot help on the growing sub-boards. Indeed, Conjecture 1 is just a special case of an important open question in the theory of hypergraph games. As we mentioned earlier, in a number of cases the probabilistic heuristic works, that is one may prove a draw for II in the $(1, 1, H)$-game, when a random argument shows the existence of a good 2-coloring of the hypergraph (X, H). It does not necessary break down when this existence of the good 2-coloring is guaranteed only by the *Lovász Local Lemma*. According to the Lovász Local Lemma there is a good 2-coloring of an even infinite hypergraph (X, H), if the maximum degree of (X, H) is "small" and the size of any $A \in H$ is "large" (see [6]). The natural direction of research is to find out if these conditions guarantee draw for the second player. Although there are very deep and promising results in [2] and [4] for the finite cases, the general solution is still far away.

Acknowledgement. Many thanks to József Beck for the lots of help and encouragement. I am also grateful to an anonymous referee for the helpful comments.

References

[1] N. Alon and J. Spencer, *The Probabilistic Method*, Academic Press, New York, (1992)

[2] J. Beck, "On positional games", *J. of Combinatorial Theory Series A* **30** (1981), 117-133.

[3] J. Beck, "Van der Waerden and Ramsey games", *Combinatorica* **1** (1981), 103-116.

[4] J. Beck, "An algorithmic approach to the Lovász Local Lemma. I.", *Random Structures and Algorithms* **2**(1991), 343-365.

[5] E.R. Berlekamp, J.H. Conway and R.K. Guy, *Winning Ways*, Volume **2**, Academic Press, New York 1982.

[6] P. Erdős and L. Lovász, "Problems and results on 3-chromatic hypergraphs and some related questions", in: *Infinite and Finite Sets eds.: A. Hajnal et al., Colloq. Math. Soc. J. Bolyai,* **11**, North-Holland, Amsterdam, 1975, 609-627.

[7] P. Erdős and J.L. Selfridge, "On a combinatorial game", *J. Combinatorial Theory Series B* **14** (1973) 298-301.

[8] M. Gardner, "Mathematical Games", *Scientific Amer.* **225**#2 (Aug.1971) 102-105; **232** #6 (June 1975) 106-111; **233** #6 (Dec. 1975) 116-119; **240** #4 (Arp. 1979) 18-28.

[9] A.W. Hales and R.I. Jewett, "Regularity and positional games", *Trans. Amer. Math. Soc.* **106**(1963) 222-229; M.R. # 1265.

[10] A. Pluhár, "Generalizations of the game k-in-a-row", *RUTCOR RESEARCH REPORT*, 15-1994.

[11] A. Pluhár, *Positional Games on the Infinite Chessboard* Ph.D. dissertation, Rutgers University 1994.

Received, February 1997

Acta Cybernetica **13** (1997) 85-102.

Evaluation Strategies of Fuzzy Datalog

Ágnes Achs*

Abstract

A fuzzy Datalog program is a set of Horn-formulae with uncertainty degrees. The meaning of a program is the fixpoints of deterministic or nondeterministic consecutive transformations. In this paper we are going to deal with the evaluation strategies of fuzzy Datalog programs. We will determine the bottom-up and top-down strategies and show their equivalence.

1 Introduction

A logical data model consists of facts and rules. The facts represent certain knowledge from which other knowledge can be deduced by the rules. In classical deductive database theory ([CGT], [U]) the Datalog-like data model is widely spread. A Datalog program is a set of Horn-clauses, that is a set of the formulae

$$A \leftarrow B_1, \ldots, B_n$$

where $A, B_i (i = 1, \ldots, n)$ are positive literals.

The meaning of a Datalog-like program is the least (if any) or a minimal model which contains the facts and satisfies the rules. This model is generally computed by a fixpoint algorithm.

In [AK2] there was given a possible extension of Datalog-like languages to fuzzy relational databases using lower bounds of degrees of uncertainty in facts and rules. This language is called fuzzy Datalog (fDATALOG). In this language the rules are completed with an implication operator and a level. We can infer the level of a rule-head from the level of the body, the level of the rule and the implication operator of the rule. We defined the deterministic and nondeterministic semantics of fDATALOG as the fixpoints of certain transformations, gave a method for fixpoint queries, and showed that this fixpoint is minimal under certain conditions.

The aim of this paper is to give some evaluation strategies of fDATALOG programs.

First we are going to summarize the concept of fDATALOG.

*Janus Pannonius University, Pollack Mihály College, Pécs, Boszorkány u. 2, Hungary, e-mail:achs@mit.pmmfk.jpte.hu

2 Basic Concepts

A *term* is a variable, constant or complex term of the form $f(t_1, \ldots, t_n)$, where f is a function symbol and $t_1, \ldots, t_n$ are terms. An *atom* is a formula of the form $p(\underline{t})$, where p is an n-arity predicate symbol and $\underline{t}$ is a sequence of terms of length n (arguments). A *literal* is either an atom (a positive literal) or the negation of an atom (a negative literal).

A term, atom, literal is *ground* if it is free of variables.

Let D be a set. The *fuzzy set* F over D is a function $F : D \to [0,1]$. Let $\mathcal{F}(D)$ denote the set of all fuzzy sets over D. So $F \in \mathcal{F}(D)$.

$$F \cup G(d) \stackrel{\text{def}}{=} \max(F(d), G(d))$$

$$F \cap G(d) \stackrel{\text{def}}{=} \min(F(d), G(d))$$

An ordering relation can be defined: $F \leq G$ iff $F(d) \leq G(d)\ \forall d \in D$. As every subset of $\mathcal{F}(D)$ has least upper bound and greatest lower bound, so $(\mathcal{F}(D), \leq)$ is a complete lattice. The top element of the lattice is $U : D \to [0,1] : U(d) = 1\ \forall d \in D$. The bottom element is: $\emptyset : D \to [0,1] : \emptyset(d) = 0\ \forall d \in D$.

Fuzzy sets are frequently denoted in the following way:

$$F = \bigcup_{d \in D} (d, \alpha_d)$$

where $(d, \alpha_d) \in D \times [0,1]$.

To make any deduction we need the concept of *implication operator*.

The features of implication operators are summarized in [DP]. In the next table we give the most frequent operators:

symbol	name	formula
$I_1(x,y)$	Gödel	1 if $x \leq y$ y otherwise
$I_2(x,y)$	Lukasiewicz	1 if $x \leq y$ $1 - x + y$ otherwise
$I_3(x,y)$	Goguen	1 if $x \leq y$ y/x otherwise
$I_4(x,y)$	Kleene-Dienes	$\max(1 - x, y)$
$I_5(x,y)$	Reichenbach	$1 - x + xy$
$I_6(x,y)$	Gaines-Rescher	1 if $x \leq y$ 0 otherwise

3 The Concept of *f*DATALOG

Definition 1 An *f*DATALOG rule is a triplet $(r; I; \beta)$, where r is a formula of the form

$$Q \leftarrow Q_0, \ldots, Q_n \quad (n \geq 0)$$

where Q is an atom (the head of the rule), $Q_1, \ldots, Q_n$ are literals (the body of the rule); I is an implication operator and $\beta \in (0, 1]$ (the level of the rule).

An *f*DATALOG rule is safe if

- All variables which occur in the head also occur in the body;
- All variables occurring in a negative literal also occur in a positive literal.

An *f*DATALOG program is a finite set of safe *f*DATALOG rules. Let A be a ground atom. The rules of the form $(A \leftarrow; I; \beta)$ are called facts.

The *Herbrand universe* of a program P (denoted by H_P) is the set of all possible ground terms constructed by using constants and function symbols occurring in P. The *Herbrand base* of P (B_P) is the set of all possible ground atoms whose predicate symbols occur in P and whose arguments are elements of H_P. A *ground instance* of a rule $(r; I; \beta)$ in P is a rule obtained from r by replacing every variable x in r by $\Phi(x)$ where Φ is a mapping from all variables occurring in r to H_P. The set of all ground instances of $(r; I; \beta)$ are denoted by $(ground(r); I; \beta)$. The ground instance of P is

$$\text{ground } (P) = \cup_{(r;I;\beta)\in P}(\text{ground } (r); I; \beta).$$

Definition 2 An interpretation of a program P, denoted by N_P, is a fuzzy set of B_P:

$$N_P \in \mathcal{F}(B_P), \text{ that is } N_P = \bigcup_{A \in B_P} (A, \alpha_A).$$

Let for ground atoms $A_1, \ldots, A_n$ $\alpha_{A_1 \wedge \ldots \wedge A_n}$ and $\alpha_{\neg A}$ be defined in the following way:

$$\alpha_{A_1 \wedge \ldots \wedge A_n} \stackrel{\text{def}}{=} \min(\alpha_{A_1}, \ldots, \alpha_{A_n})$$

$$\alpha_{\neg A} \stackrel{\text{def}}{=} 1 - \alpha_A.$$

Definition 3 An interpretation is a model of P if for each $(ground(r); I; \beta) \in ground(P)$, $ground(r) = A \leftarrow A_1, \ldots, A_n$

$$I(\alpha_{A_1 \wedge \ldots \wedge A_n}, \alpha_A) \geq \beta$$

A model M is the least model if for any model $N, M \leq N$. A model M is minimal if there is no model $N \neq M$ such that $N \leq M$.

To be short, we sometimes denote $\alpha_{A_1 \wedge \ldots \wedge A_n}$ by α_{body} and α_A by α_{head}.

The semantics of fDATALOG is defined as the fixpoints of consequence transformations. Depending on these transformations we can define two semantics for fDATALOG. The deterministic semantics is the least fixpoint of deterministic transformation, the nondeterminic semantics is the least fixpoint of nondeterministic transformation. With the aid of the deterministic transformation the rules of a program are evaluated parallely, while in nondeterministic case the rules are considered independently one after another.

These transformations are the following:

Definition 4 The consequence transformations $DT_P : \mathcal{F}(B_P) \to \mathcal{F}(B_P)$ and $NT_P : \mathcal{F}(B_P) \to \mathcal{F}(B_P)$ are defined as

$$DT_P(X) = \{\cup\{(A, \alpha_A)\} | (A \leftarrow A_1, \ldots, A_n; I; \beta) \in ground(P),$$

$$(|A_i|, \alpha_{A_i}) \in X \text{ for each } 1 \leq i \leq n,$$

$$\alpha_A = \max(0, \min\{\gamma | I(\alpha_{\text{body}}, \gamma) \geq \beta\})\} \cup X$$

and

$$NT_P(X) = \{(A, \alpha_A)\} \cup X$$

where $(A \leftarrow A_1, \ldots, A_n; I; \beta) \in ground(P), (|A_i|, \alpha_{A_i}) \in X,\ 1 \leq i \leq n,$

$$\alpha_A = \max(0, \min\{\gamma | I(\alpha_{\text{body}}, \gamma) \geq \beta\}$$

$|A|$ denotes $p(\underline{c})$ if either $A = p(\underline{c})$ or $A = \neg p(\underline{c})$ where p is a predicate symbol with arity k and $\underline{c}$ is a list of k ground terms.

We can define the powers of the transformations:

For any $T : \mathcal{F}(B_P) \to \mathcal{F}(B_P)$ transformation let

$$\begin{aligned} T_0 \quad = \quad & \{\cup\{(A, \alpha_A)\} | (A \leftarrow; I; \beta) \in ground(P), \\ & \alpha_A = \max(0, \min\{\gamma | I(1, \gamma) \geq \beta\})\} \\ & \cup\{(A, 0) | \exists (B \leftarrow \ldots \neg A \ldots, ; I; \beta) \in ground(P)\} \end{aligned}$$

and let

$$T_1 = T(T_0)$$

$$\ldots$$

$$T_n = T(T_{n-1})$$

In [AK2] it was proved, that starting from the set of facts (T_0), both DT_P and NT_P have a fixpoint, which is the least fixpoint in the case of positive P. These fixpoints are denoted by $\text{lfp}(DT_P)$ and $\text{lfp}(NT_P)$.

It was also proved, that $\text{lfp}(DT_P)$ and $\text{lfp}(NT_P)$ are models of P. These propositions are the background of the following definition:

Definition 5 We define $\text{lfp}(DT_P)$ to be the deterministic semantics and $\text{lfp}(NT_P)$ to be the nondeterministic semantics of fDATALOG programs.

For function- and negation-free fDATALOG, the two semantics are the same, but they are different if the program has any negation.

The set $\text{lfp}(DT_P)$ is not always a minimal model. In nondeterministic case, however, it is minimal under certain conditions. This condition is stratification. Stratification gives an evaluating sequence in which the negative literals are evaluated first.

To stratify a program, it is necessary to define the concept of dependency graph. This is a directed graph, whose nodes are the predicates of P. There is an arc from predicate p to predicate q if there is a rule whose body contains p or $\neg p$ and whose head predicate is q.

A program is recursive, if its dependency graph has one or more cycles.

A program is stratified if whenever there is a rule with head predicate p and a negated body literal $\neg q$, there is no path in the dependency graph from p to q.

The stratification of a program P is a partition of the predicate symbols of P into subsets $P_1, \ldots, P_n$ such that the following conditions are satisfied:

a) if $p \in P_i$ and $q \in P_j$ and there is an edge from q to p then $i \geq j$

b) if $p \in P_i$ and $q \in P_j$ and there is a rule with the head p whose body contains $\neg q$, then $i > j$.

A stratification specifies an order of evaluation. First we evaluate the rules whose head-predicates are in P_1 then those ones whose head-predicates are in P_2 and so on. The sets $P_1, \ldots, P_n$ are called the strata of the stratification.

A program P is called stratified if and only if it admits a stratification. There is a very simple method for finding a stratification for a stratified program P in [CGT],[U].

[AK2] proves that for stratified fDATALOG program P, there is an evaluation sequence, - this is the order of strata - in which $\text{lfp}(NT_P)$ is a minimal model of P.

More detailed:

Let P be a stratified fDATALOG program with stratification $P_1, \ldots, P_n$. Let P_i^* denote the set of all rules of P corresponding to stratum P_i, that is the set of all rules whose head-predicate is in P_i.

Let

$$L_1 = \text{lfp}\ (NT_{P_1^*})$$

where the starting point of the computation is the set of facts.

$$L_2 = \text{lfp}\ (NT_{P_2^*})$$

where the starting point of the computing is L_1,

$$\ldots$$

$$L_n = \text{lfp}\ (NT_{P_n^*})$$

where the starting point is L_{n-1}.

In other words: at first we compute the least fixpoint L_1, corresponding to the first stratum of P. Then one can take a step to the next stratum, and so on.

It can be seen that L_n is a minimal fixpoint of P, that is L_n = lfp (NT_P) ([AK2]).

4 Evaluation Strategies

An fDATALOG program can be evaluated with the aid of different strategies. Starting from the facts, applying the rules, all of the computable facts can be inferred, that is lfp (DT_P) or lfp (NT_P) can be determined. In this case, we speak about *bottom-up evaluation.*

In many cases however, the whole evaluation is not necessary, because we only want to get an answer to a concrete question. If a goal is specified together with an fDATALOG program, it is enough to consider only the rules and facts which are necessary to reach the goal. In the case of starting from the goal, and applying the suitable rules we infer to the facts, we speak about *top-down evaluation.*

5 Bottom-up Evaluation

For simplicity, we denote consequence transformation with T_P. This doesn't cause any trouble, because in the case of negation-free programs the fixpoints of the two tranformations are the same, and if the program contains any negation, we will consider only the nondeterministic transformation.

The fixpoint computation is a simple iteration with the following algorithm:

Algorithm 1

```
Procedure bottom_up
        old := T_0
        new := T_P(T_0)
        while old ≠ new do
                old := new
                new := T_P(old)
        endwhile
endprocedure
```

Note: In nondeterministic case, the halt condition means that none of the rules results in any new facts.

The disadvantage of the algorithm is the great number of superflouos evaluations. There are rules which are evaluated again and again in spite of the fact, that they don't result any new facts. Therefore, it is practical to omit these rules. Whether a rule can be omitted or not, depends on the path leading to the head predicate of the rule in the dependency graph. If this path contains any circle -

that is the rule is recursive - one can not omit the rule before obtaining the fix-point. But if the path does not contain any circle, then probably it can be omitted before terminating. A rule can be omitted, if the steps of the algorithm exceed the length of the maximal path leading to the head predicate of the rule. Using this observation a modified bottom-up evaluation strategy can be acquired.

6 Modified Bottom-up Evaluation

Let $P = \cup\{(r; I; \beta)\}$. Let $h : P \to N$ be defined in the following way:

$$h(r; I; \beta) = \begin{cases} n & \text{where } n \text{ is the length of the longest loopfree path leading to} \\ & \text{the headpredicate in dependency graph} \\ \infty & \text{if the path leading to the headpredicate contains any circle} \end{cases}$$

Let $T'_n = T_{P'_n}(T_{n-1})$ where

$$P'_n = P - \{(r; I; \eta) | h(r; I; \beta) < n\}$$

The sequence T'_n has a limit, that is:

Proposition 1 For function- and negation-free program P $\exists\ m \in N : T'_m = T'_{m+1} = \ldots = T'_\infty$

Proof: Let k be the number of predicates in P, n be the arguments' number of predicate with maximum argument's number and c be the number of constants in P. Then the proposition is true for $m = kc^n$.

□

For this m let T'_m be denoted with $T'(P)$.

Proposition 2 For negation- and function-free fDATALOG program P $lfp(T_P) = T'(P)$.

Proof:

a) From the construction of $T'(P)$, $T'(P) \subseteq lfp(T_P)$.

b) Let $(A, \alpha_A) \in lfp(T_P)$.

Then there is $(r; I; \beta) \in P$, for which $(A \leftarrow A_1, \ldots, A_n; I; \beta) \in ground(r)$. Let $h(r; I; \beta) = k$. Then $(r; I; \beta) \in P'_k$, so $(A, \alpha_A) \in T'_k \subseteq T'(P)$.

□

The algorithm of modified bottom-up evaluation is the following:

Algorithm 2

Procedure bottom-up 2
    $k := 1$
    old $:= T_0$
    new $:= T_P(T_0)$
    while old $\neq$ new do
        $k := k + 1$
        old := new
        $P := P - \{(r; I; \beta) | h(r; I; \beta) < k\}$
        new $:= T_P$ (old)
    endwhile
endprocedure

7 Modified Bottom-up Evaluation in the Case of Stratified fDATALOG

The modified bottom-up evaluation can be applied in the case of stratified fDATALOG. Then we can evaluate by strata. In details:

Let P be a stratified fDATALOG program with stratification $P_1, \ldots, P_n$. Let P_i^* denote the set of all rules of P corresponding to stratum P_i, that is the set of all rules whose head-predicates are in P_i.

Let

$$L_1 = lfp(NT_{P_i^*})$$

where the starting point of the computation is L_{i-1}, and $T_{P_i^*} = NT_{P_i^*} = DT_{P_i^*}$.

Because, due to the stratification of P, all negative literals of stratum i correspond to predicates of lower strata, the evaluation of P_i^* is the same as the evaluation of a negation-free program.

From this the following proposition can be made:

Proposition 3 L_i can be evaluated by the modified bottom-up evaluation, that is $L_i = T'(P_i^*)$.

8 Top-down Evaluation

In many cases we only want to get an answer to a concrete question. In such cases a goal is specified together with an fDATALOG program. Then during the evaluation it is enough to consider only the rules and facts which are necessary to reach the goal.

A *goal* is a pair $(Q\alpha)$, where Q is an atom, α is the level of the atom. It is possible, that Q contains variables, and α can be either a constant or a variable. An fDATALOG program enlarged with a goal is a *query*.

A goal can be evaluated with the aid of sub-queries. This means, that all of the rules, whose head-predicate can be unificated with the given goal-predicate are selected, and the predicates of the body are considered as new sub- goals. This procedure continues until obtaining the facts. This kind of evaluation is the top-down evaluation.

To deal with this strategy, we need some basic concepts.

Definition 6 A substitution θ is a finite set of the form $\{x_1|t_1,\ldots,x_n|t_n\}$, where $x_i(i=1,\ldots,n)$ is a distinct variable and $t_i \neq x_i$ $(i=1,\ldots,n)$ is a term. The set of variable $\{x_1,\ldots,x_n\}$ is called the domain of θ. If all terms $t_1,\ldots,t_n$ are constants, then θ is called a ground substitution. The empty substitution is denoted by ε. If θ is a substitution and t is a term, then $t\theta$ denotes the term which is defined as follows:

$$t\theta = \begin{cases} t_i & \text{if } t|t_i \in \theta \\ t & \text{otherwise.} \end{cases}$$

If L is a literal then $L\theta$ denotes the literal which is obtained from L by simultaneously replacing each variable x_i that occurs in L by the corresponding term t_i, iff $x_i|t_i$ is an element of θ.

For example, let $L = \neg p(a,x,y,b)$ and $\theta = \{x|c,y|x\}$, then $L\theta = \neg p(a,c,x,b)$.

If $(r;I;\beta)$ is a fDATALOG rule, then $(r\theta;I;\beta)$ denotes the rule, which is obtained simultaneously applying the substitution θ for all literals of r. In the body of $r\theta$ the atoms are considered with single multiplicity.

Definition 7 Let $\theta = \{x_1|t_1,\ldots,x_n|t_n\}$ and $\sigma = \{y_1|u_1,\ldots,y_n|u_n\}$ be two substitutions. The composition $\theta\sigma$ of θ and σ is obtained from the set

$$\{x_1|t_1\sigma,\ldots,x_n|t_n\sigma,y_1|u_1,\ldots,y_m|u_m\}$$

by eliminating each component of the form $z|z$ and by eliminating each component for which $y_i = x_j$ for some j.

If $(r;I;\beta)$ is a rule then applying $\theta\sigma$ to the rule has the same effect as first applying θ to r, yielding $(r\theta;I;\beta)$, and then applying σ to $r\theta$.

Definition 8 If for a pair of literals L and M a substitution θ exists, such that $L\theta = M\theta$, then we say that L and M are unifiable and the substitution θ is called a unifier. Let θ and λ be substitutions. We say that θ is more general than λ iff a substitution σ such that $\theta\sigma = \lambda$ exists.

Let L and M be two literals. A most general unifier of L and M $(mgu(L,M))$ is a unifier which is more general than any other unifier.

The concept of *mgu* has been introduced in much more general contexts, where terms may contain function symbols. There are different algorithms for determining *mgu* ([P], [U]). As now we deal with function-free fDATALOG, therefore it is practical to give a simple algorithm, which generates a *mgu* for each pair of literals L and M if they are unifiable, or tells if they are not.

Let $L = p(t_1,\ldots,t_n)$ and $M = p'(t'_1,\ldots,t'_m)$ be two literals. The function $mgu(L,M)$ can be generated in the following way:

Algorithm 3

Function $mgu(L, M)$
    if $p \neq p'$ or $n \neq m$ then L and M are not unifiable
    else
        $\theta := \varepsilon$
        $k := 1$
        unifiable := true
        while $k \leq n$ and unifiable do
            if $t_i\theta \neq t'_i\theta$
                then if $t'_i\theta$ is a variable
                    then $\theta := \theta\{t'_i\theta | t_i\theta\}$
                    else if $t_i\theta$ is a variable
                        then $\theta := \theta\{t_i\theta | t'_i\theta\}$
                        else unifiable := false endif
                    endif
            endif
            $k := k + 1$
        endwhile
        if unifiable then $mgu(L, M) = \theta$ else L and M are not unifiable
    endif
endfunction

From the algorithm one can see, that $mgu(L, M) \neq mgu(M, L)$. Because of this asymmetry we have to be very careful during the top-down evaluation.

We also need the concept of projection and join of substitutions.

Definition 9 Let $\theta = \{x_1|t_1, \ldots, x_n|t_n\}$ be substitution and let $H = \{x_{i_1}, \ldots, x_{I_k}\}$ be a set. The projection of θ to H is the substitution $\theta_H = \{x_{i_1}|t_{i_1}, \ldots, x_{i_k}|t_{i_k}\}$.

Definition 10 Let $\theta = \{x_1|t_1, \ldots, x_n|t_n\}$ and $\sigma = \{y_1|u_1, \ldots, y_n|u_n\}$ be substitutions. Let us suppose that for each pair $x_i|t_i, y_j|u_j$ for which $x_i = y_j$ is true, $t_i = u_j$ also comes true. Then the join of θ and σ is the set $\theta \otimes \sigma = \{x_1|t_1, \ldots, x_n|t_n, y_1|u_1, \ldots, y_m|u_m\}$, from which the repeated components are omitted.

If for any pair $x_i|t_i, y_j|u_j, x_i = y_j$ is true, but $t_i \neq u_j$, then the join of θ and σ is not defined.

From this definition one can see, that the join is a partial operation. If we want to apply the join and the composition together, the concept of partial composition has to be defined.

Definition 11 The partial composition of substitutions θ and σ is $\theta\sigma$, if both of them are defined and is not defined if any of substitutions is not defined.

First we deal with the evaluation of negtion-free fDATALOG programs. We will search the solution with the aid of evaluation graph. This is a special AND/OR

tree, a special hyper-graph. Every odd edge is a n-order hyper-edge with the set-node of n elements, and every even edge is an ordinary edge with one node. More precisely:

An evaluation hypergraph is a tree, whose root is the goal, the leaves are the symbols "good" and "bad", and the nodes are defined recursively.

Let the level of the root be 0. On every even level of the graph there are sub-goals, that is suitably unified heads, on every odd level there are bodies of rules.

Let Q be a node of level $k = 2i$, and let us suppose, that there are m rules in the form

$$R \leftarrow R_1, \ldots, R_n; I; \beta$$

whose heads are unifiable with Q. Then this node has m children, and these children are in the form

$$R_1\theta, \ldots, R_n\theta$$

where $\theta = mgu(Q, R)$,if $n > 0$; if $n = 0$, then the child is the symbol "good". If there are not any unifiable rule, then the child is the symbol "bad".

We have to pay attention to rename the variables, namely it is important, that the variables in the body of a unified rule let be different from the former unifications. To solve this problem, we will identify these variables by subscribing them with the level of the evaluation graph.

Let us attach labels to the edges of the form $Q \rightarrow R_1\theta, \ldots, R_n\theta$! Let the edge's label be the triplet $(\theta; I\beta)$.

Let the rule-body of the form $Q_1, \ldots, Q_n$ be a node of level $k = 2i + 1$! Then there is an n-order hyper-edge to the nodes $Q_1, \ldots, Q_n$. The hyper-edge has no label.

We can get an answer to the query from the labels of evaluating graph.

The path ending in the symbol "bad" doesn't give solution. Let us omit these paths! In other words, let us omit all of the edges and nodes which lead to this symbol independently from the fact, that these nodes are connected to each other by hyper-edges or ordinary edges. (If there is a path from one node of a hyper-edge to the symbol "bad", all of the nodes belonging to this hyper-edge and their descendants are cancelled.) The given graph is called searching graph.

A solution can be achieved along the path ending in the symbol "good" in the searching graph. The union of these solutions is the answer to the given query. The level of the atoms in the answer can be computed with the aid of the uncertainty-level function.

Definition 12 The function

$$f(I, \alpha, \beta) = \min(\{\gamma | I(\alpha, \gamma) \geq \beta\})$$

is called uncertainty-level function.

In the case of the studied implication operators $f(I, \alpha, \beta)$ is the following:

$$f(I_1, \alpha, \beta) = \min(\alpha, \beta)$$

$$f(I_2, \alpha, \beta) = \max(0, \alpha + \beta - 1)$$

$$f(I_3, \alpha, \beta) = \alpha \cdot \beta$$

$$f(I_4, \alpha, \beta) = \begin{cases} 0 & \alpha + \beta \leq 1 \\ \beta & \alpha + \beta > 1 \end{cases}$$

$$f(I_5, \alpha, \beta) = \max(0, 1 + (\beta - 1)/\alpha), \alpha \neq 0$$

$$f(I_6, \alpha, \beta) = \alpha.$$

Let us determine the substitution θ along the hyper-path leading to the symbol "good" in the following way: (As a path contains hyper-edges, therefore the path may end in more leaves.)

For each hyper-node let us construct the join of the substitutions of the body's atoms. Let us order this joins to the nodes of even levels (that is to the nodes of the heads). Then let us construct the partial composition of these substitutions.

On answer to the query

$$(Q, \alpha)$$

is:

$$(Q\theta, \alpha_{goal}),$$

where α_{goal} can be computed recursively with the aid of uncertainty-level function $f(I, \alpha, \beta)$ in the following way:

Starting at the leaves, we order to them the value $\alpha = 1$, then we go backward to the root. If the uncertainty level of a node on the odd level of the graph is α, let the uncertainty level of the parent node be $\alpha = f(I, \alpha, \beta)$, where $I; \beta$ are the values in the label of the edge. If the uncertainty level of the children of a node on the odd level of the graph is $\alpha_1, \ldots, \alpha_k$, then let the uncertainty level of the node be $\alpha = \min(\alpha_1, \ldots, \alpha_k)$. The uncertainty level of the root is α_{goal}.

Example 1 Let us see next rules:

$$p(a) \leftarrow; I_1; \beta_1$$

$$p(b) \leftarrow; I_2; \beta_2$$

$$r(c) \leftarrow; I_3; \beta_3$$

$$q(x, y) \leftarrow p(x), r(y); I_2; \beta_4$$

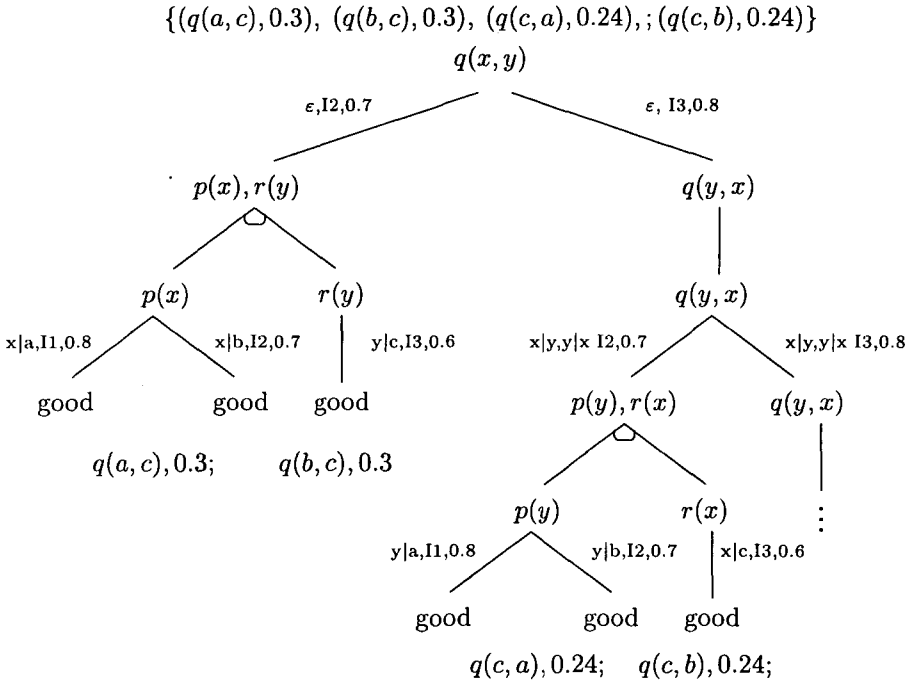
$$q(x, y) \leftarrow q(y, x); I_3; \beta_5$$

$$s(x) \leftarrow q(x, y); I_3; \beta_6$$

Let $\beta_1 = 0.8, \beta_2 = 0.7, \beta_3 = 0.6, \beta_4 = 0.7, \beta_5 = 0.8, \beta_6 = 0.9$

We want to determine $q(x, y)$.

According to the following AND/OR graph,the solution is:



It can be seen, that in the case of finite evaluation graph the bottom-up and the top-down strategy give the same result. More exactly:

Theorem 1 For a given goal and in the case of finite evaluation graph, the top-down evaluation gives the same result as the fixpont query.

Proof: We prove the equivalence of the two evaluations by induction on the depth of the evaluation graph.

Let us suppose that the depth of evaluation graph is one, that is all of the children of the root are the symbols "good" or "bad". This can occur only in the case if no rule's head can be unificated with the goal, or only facts can be unificated with that. In the first case, there is no answer to the query either in bottom-up, or top-down evaluations. In the second case, according to both of the evaluations, the answer is the same.

Let us suppose, that the theorem is true for all evaluating graphs, containing paths with length at least n.

Let us consider the evaluating graph, the maximum path-length of which is $n+1$.

Let us examine the sub-goals on the second level of the graph. The depth of the evaluation graph of these sub-goals is at least $n-1$, that is the induction assumption

is true. In bottom-up manner the goal can be reached only from these sub-goals. Going up to the first level in bottom-up manner along the hyper-edges, we get the bodies of the rules and the uncertainty level, from which we get the wanted answer. Applying the suitable substitution and computing the uncertainty factor, we get the same answer as in top-down manner.

Thus according to the induction hypothesis, the statement is true for all finite evaluation graphs. □

We give an algorithm to evaluate the graph. This algorithm provides the answer in the case of a given program and a given goal.

The algorithm consists of two procedures calling each other, one of these procedures evaluating a goal or a sub-goal, the other evaluating a rule-body.

The "goal_evalutaion" procedure determines all of the unified bodies in the case of unificable rules, and evaluating these bodies gives the answer to the goal. The "rule_evaluation" procedure evaluating the sub-goals of the body gives the substitution belonging to the body and the uncertainty level of the body.

The order of the unificable rules in the "goal_evaluation", and that of the sub-goals in the "rule_evaluation" are determined with the aid of a selection function. The special symbols ("good", "bad") are not in the set of evaluable sub-goals, because they are not evaluable. (In the case of "bad" there is no a unificable rule, in the case of "good" we get an empty node after unifying, so we can determine the answer immediately.)

It is practical to solve the join of the substitutions in top-down manner, that is not to consider the sub-goals as independent evaluations, but to narrow the size of the graph by a "sideways information passing". This means, that the substitution getting by evaluation of a sub-goal can be applied immediately to the other members of the body, so we can reduce the number of examinable paths.

During the evaluation of a sub-goal, it is possible to substitute such variables which don't appear among the variables of the sub-goal, therefore it is enough to consider only the projection of the substitution to the variables of the sub-goal.

If it is necessary, the variables can be renamed with the aid of the set of substituting-terms. The set of substituting-terms of substitution $\theta = \{x_1|t_1, \ldots, x_n|t_n\}$ is the set $\{t_1, \ldots, t_n\}$.

Algorithm 4

```
Evaluation:
begin
      solution := ∅
      goalanswer := ∅
      goal_evaluation (goal, goalanswer)
      while not_empty (goalanswer) do
            (θ, αgoal) := element (goalanswer)
            goalanswer := goalanswer −{(θ, αgoal)}
            solution := solution ∪{( goal's_atom θ, αgoal )}
      endwhile
end
```

```
Procedure goal_evaluation (goal, goalanswer)
        goal_variables := { the set of the variables of the goal }
        R := {(r; I; β)| rule's_head (r) is unifiable with the goal }
        if R = ∅ then return
        while not_empty (R) do
                (r; I; β) := rule_selection (R)
                R := R − {(r; I; β)}
                body := rule's_body (r)
                for all variable ∈ r do
                        if variable ∈ substituting_terms (θ)
                                then variable := newname (variable)
                endfor
                θ := mgu(goal's_atom, rule's_head (r))
                body := body θ
                αbody := 1
                θ body := ε
                if body = ∅ then goalanswer := goalanswer ∪{θ, f(I, αbody, β))}
                  else rule_evaluation (body, αbody, θbody, goalanswer,
                          goal_variables, I, β)
                endif
        endwhile
endprocedure

Procedure rule_evaluation (body, αbody, θbody, goalanswer, goal_variables, I, β)
        atom := atom_selection (body)
        newbody := body −{ atom }
        answer := ∅
        goal_evaluation (atom, answer)
        if answer = ∅ then return
        while not_empty (answer) do
                (θ, αatom) := element (answer)
                answer := answer −{(θ, αatom )}
                θbody := θbodyθ
                αbody := min ( αbody, αatom)
                if newbody ≠ ∅ then
                  newbody := newbody θ
                  rule_evaluation (newbody, αbody, θbody, goalanswer,
                    goal_variables, I, β)
                endif
                if newbody = ∅ then
                        θ := projection (θbody, goal_variables)
                        goalanswer := goalanswer ∪{(θ, f(I, αbody, β))}
                endif
        endwhile
endprocedure
```

Note: The order of the unificable rules and sub-goals is unimportant, but it has an effect on the efficiency of the algorithm.

The uncertainty level of the goal $(Q;\alpha)$ is either constant or variable. If it is variable, this variable gets value during the evaluation. If α is a constant, then the uncertainty level received during the execution of the algorithm is a solution only in that case, if this level is greater then α. In this case, however, it is unnecessary to consider all the rules of the program. It is enough to take those, whose uncertainty factors are greater then α. Thus, the size of the evaluation graph can be reduced.

As the above example shows, the top-down evaluation may not terminate. The reason is the evaluation of recursive atoms, because the evaluation of nonrecursive atoms terminates in finite steps. (The number of steps is $2t + 1$, where t is the longest path leading to the atom in the evaluating graph.)

If we order a depth limit to each recursive atom, the procedure can be stopped. This limit can be determined in the following way:

In a dependency graph let h be the maximum length of the loops containing the predicate of the atom, and let t be the maximum length of loop-free paths leading to this predicate.

Let us enter the concept of *recursion distance.* This is the number of steps in which we get the fixpoint respecting this atom in bottom-up evaluation. The recursion distance depends on the number of constant in the program and the "content" of the predicate.

For example in the case of the program

$$u(x,y) \leftarrow e(x,z), u(z,y); I; \beta_1$$

$$u(x,y) \leftarrow e(x,y); I; \beta_2$$

where e is a fact and c is the number of constant in the program, the recursion distance of atom $u(x,y)$ is $c-1$.

Let us denote the recursion distance by r! Then the depth limit is $k = 2h(r-1) + 2t + 1$.

Proposition 4 Let us order the previously defined depth limit to each atom of program P! Then the top-down evaluation terminates and gives all the solutions, which satisfies the goal.

Proof: As the goal-evaluation is driven back to the evaluation of the sub-goals, therefore it is enough to show the truth of the proposition for one recursive atom.

If there is no loop-free path to a rule's head-predicate in the dependency graph, the rule can not be evaluated. Thus, it is enough to look at the atoms to which there are loop-free paths.

If the length of a path leading to the predicate in the dependency graph is t, the length of this path in the evaluating graph is $2t$, because the evaluation graph is built from a series of two steps: determining the rule-bodies, and dividing them into sub-goals. There are additional edges leading to the ending symbols.

Along the $2t$ step-long path we get from the sub-goal to an atom of a fact-predicate, which can be evaluated.

The given atom - including other possible variables - occurs again in the evaluation graph in $2h$ steps deeper. The new evaluation is necessary only, if it provides a new solution. This possibility however can not occur more than the value of the recursion distance. As in the case of the first recurrence, we are on the second recursion level, so it is enough to allow the recurrence in $r - 1$ times.

As we don't get any new solution deeper then the given limit, we can leave these branches.

Ordering the suitable depth limit to each atom, the algorithm terminates, and gives all the solutions which satisfies the goal. □

Note: The recursion distance is not always as simple as in the example above, but it can not be greater then c^n, where c is the number of constants, n is the number of atoms in the program.

9 Top-down Evaluation in the Case of Stratified *f*DATALOG

It is easy to apply the top-down evaluation for stratified *f*DATALOG. In the case of stratified *f*DATALOG, the head-predicate of a rule is at least as high stratum as the predicates of the body. In other words, during the top-down evaluation we approach from the higher strata to the lower ones, that is in the evaluation graph the stratum of a parent node is not lower than the stratum of the children. Therefore when we compute the uncertainty level, we are starting at the lowest stratum. This observation can be used to handle the negated predicates. If a sub-goal is negated, let us indicate this sub-goal, and pay attention to this marking during the computation of the uncertainty level. If the atom is marked and the uncertainty level computed up to this point is α, let us continue the computation with value $1 - \alpha$.

10 Conclusion

In this article we have dealt with the evaluation of fuzzy DATALOG, given the algorithms of bottom-up and top-down evaluation, and showed the equivalence of two evaluations.

References

[AK1] Ágnes Achs, Attila Kiss: Fixpoint query in fuzzy Datalog, *Annales Univ. Sci. Budapest, Sect. Comp.* 15 (1995) 223-231

[AK2] Ágnes Achs, Attila Kiss: Fuzzy Extension of Datalog, *Acta Cybernetica* 12 (1995) 153-166.

[CGT] S. Ceri G. Gottlob L. Tanca: *Logic Programming and Databases*, Springer-Verlag Berlin, 1990

[DP] Didier Dubois - Henri Prade: Fuzzy sets in approximate reasoning, Part 1: Inference with possibility distributions, *Fuzzy Sets and Systems* 40 (1991) 143-202.

[GS] Yuri Gurevich, Saharon Shelah: Fixed-point extensions of first-order logic, *IEEEE Symp. on FOCS* (1985), 346-353.

[K] Attila Kiss: On the least models of fuzzy Datalog programs, *International Conference on Information Processing and Management of Uncertainty in Knowledge-based system*, Mallorca 465-471.

[L] J. W. Lloyd: *Foundations of Logic Programming*, Springer-Verlag, Berlin, 1987.

[LL] Deyi Li, Dongbo Liu: *A Fuzzy PROLOG Database System*, Research Studies Press LTD., Taunton, Somerset, England, 1990.

[N] Vilém Novák: *Fuzzy sets and their applications*, Adam Hilger Bristol and Philadelphia, 1987.

[P] Pásztorné Varga Katalin: *A matematikai logika és alkalmazásai* Tankönyvkiadó, Bp., 1986.

[U] J.D. Ullman: *Principles of database and knowledge-base systems*, Computer Science Press, Rockville, 1988.

Received July, 1996

CONTENTS

ISSN 0324—721 X

Felelős szerkesztő és kiadó: Csirik János
A kézirat a nyomdába érkezett: 1997. szeptember
Terjedelem: 7,12 (B/5) ív