tanulmányok

208/1988

MTA Számitástechnikai és Automatizálási Kutató Intézet Budapest

Magyar Tudományos Akadémia
Számítástechnikai és Automatizálási Kutató Intézete
Computer and Automation Institute, Hungarian Academy of Sciences

PROCEEDINGS OF THE

5TH INTERNATIONAL MEETING OF YOUNG COMPUTER SCIENTISTS

(IMYCS'88)

Smolenice Castle, Czechoslovakia, November 14-18, 1988

Edited by

E. Csuhaj-Varjú and *J. Demetrovics*

Computer and Automation Institute, Hungarian Academy of Sciences
Budapest

and

J. Kelemen

Institute of Computer Science, Comenius University
Bratislava

TANULMÁNYOK 208/1988

STUDIES 208/1988

A kiadásért felelős:

KEVICZKY LÁSZLÓ

Főosztályvezető:

DEMETROVICS JÁNOS

ISBN 963 311 250 8

ISSN 0324-2951

PREFACE

This volume contains the texts of the invited lectures and short communications presented at the **5th International Meeting of Young Computer Scientists** held at Smolenice Castle, Czechoslovakia, November 14-18, 1988.

The meeting was organized by the **Association of the Slovak Mathematicians and Physicists** in cooperation with the **Computer and Automation Institute** of the **Hungarian Academy of Sciences**, Budapest and the **Institute of Computer Science** of the **Comenius University**, Bratislava. The aim of the meeting was to promote research beginners in computer science, to focus their professional attention to some distinguished problems, and to create an opportunity for establishing professional relations.

The twenty short communications included in the program of IMYCS'88 was selected from about 50 submissions. All the published texts have been completed in camera-ready form by the authors.

We wish to express our gratitude to E. Csuhaj-Varjú (Budapest), S. K. Dulin (Moscow), J. Karhumaki (Turku), A. Kelemenová (Bratislava), J. Sakarovitch (Paris) and M. Szijártó (Győr) for their active participation in the work of the Program Committee of IMYCS'88, as well as to P. Borovansky, R. Creutzburg, V. Dobrovolny, P. Duris, H. Harz, J. Hromkovic, P. Kaiser, I. Kalas, J. Kelemen, M. Královά, P. Mikulecky, J. Niznansky, H. Reichel, P. Ruzicka, L. Staiger, J. Sturc, O. Sykora, J. Vogel, J. Vrto, G. Wechsung and J. Wiedermann for their cooperation with the Program Committee as subreferees.

Special thanks go to J. Dassow (Magdeburg) for chairing the Program Committee of IMYCS'88 and to the **Computer and Automation Institute** of the **Hungarian Academy of Sciences** for publishing the proceedings of the IMYCS'88.

Budapest and Bratislava, May 1988

E. Csuhaj-Varjú,
J. Demetrovics,
J. Kelemen

TABLE OF CONTENTS

INVITED LECTURES

IMYCS'88, Smolenice Castle, November 14-18, 1988.

AN EXTENSION OF THE KROHN-RHODES DECOMPOSITION OF AUTOMATA

Zoltán Ésik

Bolyai Institute, A. József University
Aradi v. tere 1, Szeged, 6720, Hungary

Abstract. The notion of an irreducible semigroup has been fundamental to the Krohn-Rhodes decomposition. In this paper we study a similar concept and point out its equivalence with the Krohn-Rhodes irreducibility. We then use the new aspect of irreducible semigroups to provide cascade decompositions of automata in a situation when a strict letter-to-letter replacement is essential. The results are stated in terms of completeness theorems. Our terminology follows [10], so that the cascade composition is referred to as the α_0-product.

1. BASIC NOTIONS

For a finite nonempty set X, let X* denote the free monoid of all *words* over X, including the *empty word* λ. We set $X^+ = X^*-\{\lambda\}$ and $X^\lambda = X \cup \{\lambda\}$.

An *automaton* is a triple $A = (A,X,\delta)$ with finite nonempty

sets A (*states*), X (*input letters*) and *transition* $\delta: A\times X\to A$. The function δ extends to a map $A\times X^*\to A$ as usual. Given a word $u\in X^*$, define the mapping $u_{\mathbb{A}}: A\to A$ by $au_{\mathbb{A}} = \delta(a,u)$, for all $a\in A$. We set $S_1(\mathbb{A}) = \{u_{\mathbb{A}} : u\in X^*\}$ and $S(\mathbb{A}) = \{u_{\mathbb{A}} : u\in X^+\}$. $S_1(\mathbb{A})$ is called the *characteristic monoid* of $\mathbb{A}$, while $S(\mathbb{A})$ is the semigroup of $\mathbb{A}$.

Our fundamental notion is the α_0-product of automata. Let $\mathbb{A}_t = (A_t, X_t, \delta_t)$, $t = 1,\ldots,n$, $n\geq 0$, be automata. For each t, let $\phi_t: A_1\times\ldots\times A_{t-1}\times X\to X_t^*$ be a (*feedback*) function, where X is a new finite nonempty set. The α_0^*-*product* $\mathbb{A} = \mathbb{A}_1\times\ldots\times\mathbb{A}_n(X,\phi)$ is defined to be the automaton (A,X,δ), where $A = A_1\times\ldots\times A_n$ and

$$\delta((a_1,\ldots,a_n),x) = (\delta_1(a_1,u_1),\ldots,\delta_n(a_n,u_n)),$$
$$u_t = \phi_t(a_1,\ldots,a_n,x),\quad t = 1,\ldots,n,$$

for all $(a_1,\ldots,a_n)\in A$ and $x\in X$. In the special case that each ϕ_t maps into $X_t^+ - (X_t^\lambda, X_t)$, we obtain the notion of the α_0^+-*product* (α_0^*-product, α_0-product). Let K be any class of automata. We define:

$\mathfrak{P}_0^*(K)$:= all α_0^*-products of automata from K,
$\mathfrak{H}(K)$:= all homomorphic images of automata from K,
$\mathfrak{S}(K)$:= all subautomata of automata from K.

The operators $\mathfrak{P}_0^+, \mathfrak{P}_0^\lambda$ and $\mathfrak{P}_0$ are defined likewise and correspond to the formations of α_0^+-products, α_0^λ-products and α_0-products. In this paper the main object of study is the combination $\mathfrak{HSP}$, where $\mathfrak{P}$ is any of the above product operators.

As defined here, the α_0-product is obtained as a special case of each of the following: α_0^*-product, α_0^+-product and α_0^λ-product. Moreover, any α_0^+-product or α_0^λ-product is an

α_0^*-product. It is however important to note that the converse also holds. For an automaton $\mathbb{A} = (A,X,\delta)$ define $\mathbb{A}^* = (A,S_1(\mathbb{A}),\delta^*)$ with $\delta^*(a,u_{\mathbb{A}}) = au_{\mathbb{A}}$, for all $a\in A$ and $u\in X^*$. Similarly, let $\mathbb{A}^+ = (A,S(\mathbb{A}),\delta^+)$ and $\mathbb{A}^\lambda = (A,\{x_{\mathbb{A}}:x\in X^\lambda\},\delta^\lambda)$, where $\delta^+(a,u_{\mathbb{A}}) = \delta(a,u)$ and $\delta^\lambda(a,x_{\mathbb{A}}) = \delta(a,x)$ for every $a\in A$, $u\in X^+$ and $x\in X^\lambda$. If K is a class of automata and z is any modifier $*$, $+$ or λ, then we have $\mathbb{P}_0^z(K) = \mathbb{P}_0(K^z)$, so that the α_0^z-product can be defined in terms of the α_0-product.

The α_0-product is equivalent to either one of the following: *loop-free product* [12], *series-parallel composition* [1], *cascade composition* [1,11]. Our terminology follows [10]. The index 0 indicates that the α_0-product is the bottom of a hierarchy connecting the loop-free product to the *Gluškov-type product*. The hierarchy of α_i-products is the subject of [10]. The automaton $\mathbb{A}^*$ corresponds to the *transformation monoid* of an automaton $\mathbb{A}$ and $\mathbb{A}^+$ is just the *transformation semigroup* of $\mathbb{A}$. The operators $\mathbb{P}^*_{\alpha_0}$ and $\mathbb{P}^+_{\alpha_0}$ thus correspond to the *wreath product* of transformation semigroups and/or monoids, see [5].

2. COMPLETENESS

The Krohn-Rhodes Decomposition Theorem, that we recall below, is a basis for studying the α_0-product. But first we need some definitions.

Let S and T be (finite) semigroups. It is said that S *divides* T, written $S<T$, if and only if S is a homomorphic image of a subsemigroup of T. Following [1], a semigroup S is called *irreducible*, if for every nonempty class K and automaton

$A \in H\$P_0(K)$, the condition $S<S(A)$ implies that $S<S(B)$ for some $B \in K$. As in [1], by U_3 we denote the monoid with two right zero elements. The divisors of U_3 are the trivial semigroup U_0, the two-element monoid with a right zero element U_1 and the two element right zero semigroup U_2. The semigroups U_i, $i = 0,1,2,3$, are called *units*. Recall that a group G is *simple* if it has no nontrivial proper normal subgroup.

Let S be a semigroup and S^1 the smallest monoid containing S as a subsemigroup. We define $Aut(S) = (S^1, S, \delta)$ with $\delta(s,t) = st$, for all $s \in S^1$ and $t \in S$. If $\mathcal{S}$ is a class of semigroups then let $Aut(\mathcal{S}) = \{Aut(S) : S \in \mathcal{S}\}$.

A *permutation automaton* is an automaton A such that $S_1(A)$ is a group. Equivalently, $A = (A,X,\delta)$ is a permutation automaton if and only if x_A is a permutation for each $x \in X$. A *discrete automaton* is an automaton as above with x_A the identical mapping $A \to A$ for each $x \in X$.

Theorem 1. *Krohn-Rhodes Decomposition Theorem.* (i) Let A be an automaton and $\mathcal{G}$ the class of those simple groups G with $G<S(A)$. Then $A \in H\$P_0(Aut(\mathcal{G} \cup \{U_3\}))$. If A is a permutation automaton which is not discrete, then $A \in H\$P_0(Aut(\mathcal{G}))$.

(ii) The irreducible semigroups are the simple groups and the units.

Let $\mathcal{K}$ and $\mathcal{K}_0$ be two classes of automata and take any variant of the α_0-product. Let $\mathbb{P}$ be the corresponding product operator. We say that $\mathcal{K}_0$ is α_0-*complete* (α_0^*-*complete*, ...) for $\mathcal{K}$

if $K \subseteq \mathfrak{HSP}(K_0)$. In particular, an α_0-complete (α_0^*-complete,...) class for the class of all automata is called an α_0-complete (α_0^*-complete,...) class.

Let G be a nonempty class of simple groups. For $i = 0,1,2,3$, define $K_i(G) = \mathfrak{HSP}_0(\mathrm{Aut}(G \cup \{U_i\}))$ and $K_{12}(G) = \mathfrak{HSP}_0(\mathrm{Aut}(G \cup \{U_1, U_2\}))$. To avoid trivial situations, when writing $K_0(G)$, we shall alway assume that G contains a non-trivial simple group.

Corollary 2. A class K of automata is α_0^+-complete (α_0^*-complete) for $K_i(G)$, $i = 0,1,2,3$, if and only if the following hold:

(i) For every $G \in G$ there is $A \in K$ with $G < S(A)$ ($G < S_1(A)$).

(ii) There is an automaton $A \in K$ with $U_i < S(A)$ ($U_i < S_1(A)$).

K is α_0^+-complete (α_0^*-complete) for $K_{12}(G)$ if and only if K satisfies (i) and (ii) with $i = 1,2$.

Notice that the conditions $G < S(A)$ and $G < S_1(A)$ are equivalent for any group G and automaton A. For various formalizations and proofs of the Krohn-Rhodes Decomposition Theorem and Corollary 2, see [1,5,10,11,13]. By the Krohn-Rhodes Decomposition Theorem, an automaton A belongs to $K_3(G)$ if and only if, for every simple group G with $G < S(A)$ we have $G < H$ for some $H \in G$. The class $K_0(G)$ consists of all permutation automata in $K_3(G)$. For further characterizations see [5], the references contained in [5], as well as [14,15]. When G is the class of all simple groups, $K_3(G)$ is the class of all automata.

Corollary 3. A class K is α_0^+-complete (α_0^*-complete) if and only if the following hold:

(i) For every (simple) group G there is $A \in K$ with $G < S(A)$.

(ii) There is $A \in K$ with $U_3 < S(A)$ ($U_3 < S_1(A)$).

The conditions involved in Corollaries 2 and 3 are only necessary for α_0-completeness. For some particular cases, necessary and sufficient conditions were obtained in [4,7,8]. The following concept was first suggested in [6] and further examined in [3,7]. Let S be a semigroup and $A = (A,X,\delta)$ and automaton. Put $S \mid^{(n)} S(A)$ for an integer $n \geq 1$ if and only if there exist a subsemigroup T of $S(A)$ and an onto homomorphism $\Psi: T \to S$ such that $\Psi^{-1}(s) \cap \{u_A : u \in X^n\} \neq \emptyset$, for all $s \in S$. Here X^n denotes the set of all words over X with length n. We say that S *divides* $S(A)$ *in equal length*, denoted $S \mid S(A)$, if and only if, $S \mid^{(n)} S(A)$ for some n.

The $\mid$-irreducible semigroups are now defined in the same way as irreducible semigroups. A semigroup S is said to be $\mid$-*irreducible* if and only if, for every nonempty K and $A \in H\mathfrak{S}\mathfrak{P}_0(K)$, $S \mid S(A)$ implies the existence of an automaton $B \in K$ with $S \mid S(B)$.

Theorem 4. [7] A semigroup is $\mid$-irreducible if and only if it is irreducible.

By a *counter* we mean an automaton $\mathfrak{C}_n = (\{a_0,\ldots,a_{n-1}\}, \{x\},\delta)$, where $n \geq 1$ and $\delta(a_i,x) = a_{i+1 \bmod n}$.

Theorem 5. [3] If $S|^{(n)}S(\mathbb{A})$ then $\mathrm{Aut}(S) \in H\$P_0(\{\mathbb{C}_n, \mathrm{Aut}(U_2), \mathbb{A}\})$.

An automaton $\mathbb{A} = (A,X,\delta)$ is called *strongly connected* if for each pair of states $a,b \in A$ there is a word $u \in X^*$ with $\delta(a,u) = b$. Moreover, $\mathbb{A}$ is *unambigous* if and only if $\delta(a,x) = \delta(a,y)$ for all $a \in A$ and $x,y \in X$. Otherwise $\mathbb{A}$ is called *umbigous*. Using Theorems 4 and 5, the following results can be proved:

Theorem 6. [7] Let G be a nonempty class of simple groups and K a class of automata such that $H\$P_0(K)$ contains the counters. Assume the following, where $i = 2$ or $i = 3$:

(i) For every $G \in G$ there is $\mathbb{A} \in K$ with $G|S(\mathbb{A})$.

(ii) There is $\mathbb{A} \in K$ with $U_i|S(\mathbb{A})$.

(iii) $H\$P_0(K)$ contains a strongly connected umbigous automaton.

Then K is α_0-complete for $K_i(G)$. Assuming (i), (iii) and (ii) for $i = 1$ and $i = 2$, it follows that K is α_0-complete for $K_{12}(G)$.

Theorem 7. [7] Let G be a class of simple groups that contains the groups of prime order. A class K is α_0-complete for $K_i(G)$, $i = 2,3$, if and only if the three conditions below hold:

(i) For every $G \in G$ there is $\mathbb{A} \in K$ with $G|S(\mathbb{A})$.

(ii) There is $\mathbb{A} \in K$ with $U_i|S(\mathbb{A})$.

(iii) $H\$P_0(K)$ contains the counters and at least one

strongly connected ambigous automaton.

Moreover, K is α_0-complete for $K_{12}(G)$ if and only if each of (i), (ii) with i = 1,2 and (iii) holds.

It should be noted that for a nonabelian simple group G and an automaton A, the two conditions $G<S(A)$ and $G|S(A)$ are equivalent. This follows from a strong result in [2], see also [7] for a direct proof. Thus, (i) of Theorem 6 or 7 can be devided into two parts: (i_1) For every nonabelian $G\in\mathcal{G}$ there is $A\in K$ with $G<S(A)$; (i_2) For every abelian $G\in\mathcal{G}$ there is $A\in K$ with $G|S(A)$. On the other hand, it is obvious that $U_i<S(A)$ if and only if $U_i|S(A)$, for each unit semigroup. Thus we can replace the condition $U_i|S(A)$ by $U_i<S(A)$ in Theorem 6 and Theorem 7. Taking into account the above remarks and the fact that each group is embedded in a nonabelian simple group, Theorem 7 readily implies the following characterization of α_0-complete classes that strengthens the main result of [8]:

Corollary 8. [3] A class K is α_0-complete if and only if the following are true:

(i) For every (simple) group G there is $A\in K$ with $G<S(A)$.

(ii) There is $A\in K$ with $U_3<S(A)$.

(iii) $H\mathfrak{S}P_{\alpha_0}(K)$ contains the counters and at least one strongly connected ambigous automaton.

Theorem 7 and Corollary 8 are in a sense the best poss-

ible results. Here we point out this fact only for Corollary 8, for Theorem 7 see [7]. Let us call a class K_0 *critical* if for every K, (i) and (ii) of Corollary 8 together with the stipulation $K_0 \subseteq H\$P_0(K)$ imply that K is α_0-complete.

Theorem 9. [4] A class K_0 is critical if and only if $H\$P_0(K_0)$ contains the counters and at least one strongly connected ambigous automaton.

We now turn our attention to the α_0^λ-product. Note that $K_2(G) \subseteq H\$P_{\alpha_0}^{\lambda}(K)$ is and only if $K_3(G) \subseteq H\$P_0^\lambda(K)$.

Theorem 10. [9] Let G be a nonempty class of simple groups and K any class of automata. K is α_0^λ-complete for $K_3(G)$ if and only if the following hold:

(i) For every $G \in G$ there is $A \in K$ with $G < S(A)$.

(ii) There is $A \in K$ with $U_3 < S_1(A)$.

(iii) K is not counter-free.

Here the last condition means that K contains an automaton (A,X,δ), which has distinct states $a_0,\ldots,a_{n-1}$, $n \geq 2$, and an input letter x with $\delta(a_i,x) = a_{i+1 \bmod n}$, for all $i = 0,\ldots,n-1$. It is easily seen that K is not counter free if and only if there is a nontrivial counter in $H\$P_0(K)$. It should be noted that, in [9], Theorem 10 is stated in a somewhat weaker form.

Corollary 11. [9] A class K is α_0^λ-complete if and only if the following are true:

(i) For every (simple) group G there is $\mathbb{A} \in K$ with $G < S(\mathbb{A})$.

(ii) There is $\mathbb{A} \in K$ with $U_3 < S_1(\mathbb{A})$.

(iii) K is not counter-free.

3. VARIETIES

Let K be a class of automata. If K is closed under the formation of α_0-products, subautomata and homomorphic images, then K is called an α_0-*variety*. Similarly, for $z = *, +, \lambda$, an α_0^z-*variety* is a class K satisfying $\mathbb{P}_0^z(K) \subseteq K$, $\mathbb{S}(K) \subseteq K$ and $\mathbb{H}(K) \subseteq K$. It is known that for each class K, $\mathbb{HSP}_0(K)$ is the smallest α_0-variety including K. Analogous fact is true for α_0^z-varieties. It follows from our definition that each α_0^*-variety is an α_0^+-variety and also an α_0^λ-variety, furthermore, α_0^+-varieties and α_0^λ-varieties are α_0-varieties. The converse direction fails, yet it holds that every 'large' α_0-variety is an α_0^+-variety. By Z_p, where p is a prime number, we denote a cyclic group of order p.

Theorem 12. [6] Every α_0-variety containing $\mathrm{Aut}(U_2)$ and each automaton $\mathrm{Aut}(Z_p)$, where p is any prime, is an α_0^+-variety.

The essence of Theorem 12 is that there is a bijective correspondence between 'large' α_0-varieties and 'large' *closed classes* of transformation semigroups in the sense of [5]. The

α_0-variety $V_0 = H\$P_{\alpha_0}(\mathrm{Aut}(\{U_2, Z_p : p \text{ is prime}\}))$ is just the class $K_2(G)$ with G consisting of the cyclic groups of prime order. Moreover, V_0 is the class of all automata that could be called *locally solvable*, see [14,15].

Corollary 13. [6] Every α_0-variety containing the automaton $\mathrm{Aut}(U_3)$ and all the automata $\mathrm{Aut}(Z_p)$ for prime numbers p is an α_0^*-variety.

Note that each α_0-variety V with $\mathrm{Aut}(U_3) \in V$ and $\mathrm{Aut}(Z_p) \in V$ for each prime p is of the form $K_3(G)$ with G containing the abelian simple groups. The smallest such α_0-variety is identified as the class of *solvable automata*.

Theorem 14. [6] If an α_0^λ-variety contains $\mathrm{Aut}(U_3)$ and a nontrivial counter, then it is an α_0^*-variety.

REFERENCES

[1] Arbib, M. A. (Ed.), Algebraic Theory of Machines, Languages, and Semigroups, with a major contribution by K. Krohn and J. L. Rhodes (Academic Press, New York, 1968).

[2] Dénes, J. and P. Hermann, On the product of all elements in a finite group, Ann. of Discrete Mathematics, 15(1982) 107-111.

[3] Dömösi, P. and Z. Ésik, On homomorphic realization of automata with α_0-products, Papers on Automata Theory, VIII(1968) 63-97.

[4] Dömösi, p. and Z. Ésik, Critical classes for the α_0-product, Theoret. Comput. Sci., to appear.

[5] Eilenberg, S., Automata, Languages, and Machines, vol. B (Academic Press, New York, 1976).

[6] Ésik, Z., Varieties of automata and transformation semigroups, submitted.

[7] Ésik, Z., Results on homomorphic realization of automata by α_0-products, in preparation.

[8] Ésik, Z. and P. Dömösi, Complete classes of automata for the α_0-product, Theoret. Comput. Sci., 47(1986) 1-14.

[9] Ésik, Z. and J. Virágh, On products of automata with identity, Acta Cybernetica, 7(1986) 299-311.

[10] Gécseg, F., Products of Automata (Springer-Verlag, Berlin, 1986).

[11] Ginzburg, A., Algebraic Theory of Automata (Academic Press, New York, 1968).

[12] Hartmanis, J. and R. E. Stearns, Algebraic Structure Theory of Sequential Machines (Prentice-Hall, Englewood Cliffs, 1966).

[13] Lallement, G., Semigroups and Combinatorial Applications (John Wiley, New York, 1979).

[14] Straubing, H., Finite Semigroup varieties of the form V*D, J. of Pure and Appl. Alg., 36(1985) 53-94.

[15] Thérien, D. and A. Weiss, Graph congurences and wreath products, J. of Pure and Appl. Alg., 36(1985) 205-215.

IMYCS'88, Smolenice Castle, November 14-18, 1988.

A Survey of Two-Dimensional Automata Theory

Katsushi Inoue and Itsuo Takanami

Department of Electronics
Faculty of Engineering
Yamaguchi University
Ube, 755 Japan

Abstract. The main purpose of this paper is to survey several properties of alternating, nondeterministic, and deterministic two-dimensional Turing machines (including two-dimensional finite automata and marker automata), and to briefly survey cellular types of two-dimensional automata.

1. Introduction

During the past thirty years, many investigations about automata on a one-dimensional tape (i.e., string) have been made (for example, see [25]). On the other hand, since Blum and Hewitt [3] studied two-dimensional finite automata and marker automata, several researchers have been investigating a lot of properties about automata on a two-dimensional tape .

The main purpose of this paper is to survey main results of two-dimensional sequential automata obtained since [3], and to give several open problems. Chapter 2 concerns alternating, nondeterministic, and deterministic two-dimensional Turing machines (including finite automata and marker automata). Section 2.1 gives preliminaries necessary for the subsequent discussions. Section 2.2 gives a difference among alternating, nondeterministic, and deterministic machines. Section 2.3 gives a difference between three-way and four-way machines. Section 2.4 states space complexity results of two-dimensional Turing machines. Sections 2.5 and 2.6 states closure properties and decision problems, respectively. Section 2.7 concerns recognition of connected pictures. Section 2.8 states other topics. Chapter 3 briefly surveys cellular types of two-dimensional automata.

2. Alternating, Nondeterministic, and Deterministic Turing Machines

This chapter concerns alternating, nondeterministic, and deterministic two-dimensional Turing machines, including two-dimensional finite automata and marker automata.

2.1. Preliminaries

Let Σ be a finite set of symbols. A two-dimensional tape over Σ is a two-dimensional rectangular array of elements of Σ. The set of all two-dimensional tapes over Σ is denoted by $\Sigma^{(2)}$.

For a tape $x \in \Sigma^{(2)}$, we let $\ell_1(x)$ be the number of rows of x and $\ell_2(x)$ be the number of columns of x. If $1 \le i \le \ell_1(x)$ and $1 \le j \le \ell_2(x)$, we let $x(i,j)$ denote the symbol in x with coordinates (i,j). Furthermore, we define

$$x[(i,j),(i',j')],$$

when $1 \le i \le i' \le \ell_1(x)$ and $1 \le j \le j' \le \ell_2(x)$, as the two-dimensional tape z satisfying the following: (i) $\ell_1(z) = i'-i+1$ and $\ell_2(z) = j'-j+1$, (ii) for each k,r $[1 \le k \le \ell_1(z), 1 \le r \le \ell_2(z)]$, $z(k,r) = x(k+i-1, r+j-1)$.

We now give some definitions of two-dimensional alternating Turing machines.

Definition 2.1. A two-dimensional alternating Turing machine (ATM) is a seven-tuple $M=(Q,q_0,U,F,\Sigma,\Gamma,\delta)$, where (1) Q is a finite set of states, (2) $q_0 \in Q$ is the initial state, (3) $U \subseteq Q$ is the set of universal states, (4) $F \subseteq Q$ is the set of accepting states, (5) Σ is a finite input alphabet ($\# \notin \Sigma$ is the boundary symbol), (6) Γ is a finite storage tape alphabet ($B \in \Gamma$ is the blank symbol), and (7) $\delta \subseteq (Q \times (\Sigma \cup \{\#\}) \times \Gamma) \times (Q \times (\Gamma - \{B\}) \times \{\text{left,right,up,down,no move}\} \times \{\text{left,right,no move}\})$ is the next move relation.

A state q in Q-U is said to be existential. As shown in Fig.1, the machine M has a read-only rectangular input tape with boundary symbols "#" and one semi-infinite storage tape, initially blank. Of course, M has a finite control, an input head, and a storage tape head. A position is assigned to each cell of the storage tape, as shown in Fig.1. A step of M consists of reading one symbol from each tape, writing a symbol on the storage tape, moving the input and storage heads in specified directions (left,right,up,down,or no move for input head, and left,right, or no move for storage head),and entering a new state, in accordance with the next move relation δ.

A configuration of an ATM $M=(Q,q_0,U,F,\Sigma,\Gamma,\delta)$ is an element of $\Sigma^{(2)} \times (N \cup \{0\})^2 \times S_M$, where $S_M = Q \times (\Gamma - \{B\})^* \times N$, and N denotes the set of all positive integers. The first component x of a configuration $c=(x,(i,j),(q,\alpha,k))$ represents the input to M. The second component (i,j) of c represents the input head position. The third component (q,α,k) of c represents the state of the finite control, nonblank contents of the storage tape, and the storage-head position. If q is the state associated with configuration c, then c is said to be universal (existential, accepting) configuration if q is a universal (existential, accepting) state. The initial configuration of M on input x is $I_M(x) = (x,(1,1),(q_0,\lambda,1))$, where λ denotes the empty string. We write $c \vdash_M c'$ and say c' is a successor of c if configuration c' follows from configuration c in one step of M, according to the transition rules δ. A computation tree of M is a finite, nonempty labeled tree with the properties,

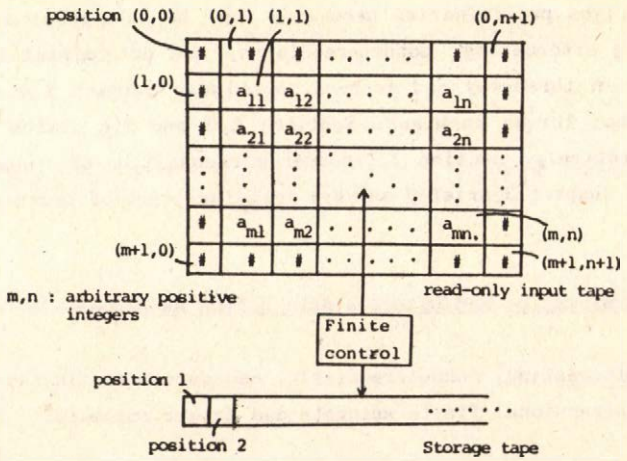


Fig. 1. Two-dimensional alternating Turing machine.

(1) each node π of the tree is labeled with a configuration $\mathcal{Q}(\pi)$,

(2) if π is an internal node (a nonleaf) of the tree, $\mathcal{Q}(\pi)$ is universal, and

$$\{c \mid \mathcal{Q}(\pi) \vdash_M c\}=\{c_1,\ldots,c_k\},$$

then π has exactly k children $\rho_1,\ldots,\rho_k$ such that $\mathcal{Q}(\rho_i)=c_i$,

(3) if π is an internal node of the tree and $\mathcal{Q}(\pi)$ is existential, then π has exactly one child ρ such that $\mathcal{Q}(\pi) \vdash_M \mathcal{Q}(\rho)$.

An <u>accepting computation tree</u> of M on x is a computation tree whose root is labeled with $I_M(x)$ and whose leaves are all labeled with accepting configurations. We say that M <u>accepts</u> x if there is an accepting computation tree of M on input x. Define $T(M)=\{x\in\Sigma^{(2)} \mid M \text{ accepts } x\}$.

A <u>three-way two-dimensional alternating Turing machine</u> (TATM) is an ATM whose input head can move left, right, or down, but not up.

A <u>two-dimensional nondeterministic Turing machine</u> (NTM) (a <u>three-way two-dimensional nondeterministic Turing machine</u> (TNTM)) is an ATM (TATM) which has no universal state. A <u>two-dimensional deterministic Turing machine</u> (DTM) (a <u>three-way two-dimensional deterministic Turing machine</u> (TDTM)) is an ATM (TATM) whose configurations each have at most one successor.

Let $L(m,n):N^2\rightarrow R$ be a function with two variables m and n, where R denotes all non-negative real numbers. With each ATM (TATM,NTM,TNTM,DTM,TDTM) M we associate a space complexity function SPACE which takes configuration $c=(x,(i,j),(q,\alpha,k))$ to natural numbers. Let SPACE(c)=the length of α. We say that M is <u>L(m,n) space-bounded</u> if for all $m,n\geq 1$ and for all x with $\mathcal{Q}_1(x)=m$ and $\mathcal{Q}_2(x)=n$, if x is accepted by M, then there is an accepting computation tree of M on input x such that, for each node π of the tree, $SPACE(\mathcal{Q}(\pi))\leq \lceil L(m,n)\rceil$. By "ATM(L(m,n))" ("TATM(L(m,n))", "NTM(L(m,n))", "TNTM(L(m,n))", "DTM(L(m,n))", "TDTM(L(m,n))") we denote an L(m,n) space bounded ATM (TATM, NTM, TNTM, DTM, TDTM).

We are also interested in two-dimensional Turing machines M whose input tapes are restricted to square ones. Let $L(m):N\rightarrow R$ be a function with one variable m. We say that M is <u>L(m) space-bounded</u> if for all $m\geq 1$ and for all x with $\mathcal{Q}_1(x)=\mathcal{Q}_2(x)=m$, if x is accepted by M, then there is an accepting computation tree of M on x such that, for each node π of the tree, $SPACE(\mathcal{Q}(\pi))\leq L(m)$. By "ATM^s(L(m))" ("TATM^s(L(m))", "NTM^s(L(m))", "TNTM^s(L(m))", "DTM^s(L(m))", "TDTM^s(L(m))") we denote an L(m) space-bounded ATM (TATM, NTM, TNTM, DTM, TDTM) whose input tapes are restricted to square ones.

For any constant $k\geq 0$, a k space-bounded ATM (NTM, DTM) is called a <u>two-dimensional alternating (nondeterministic, deterministic) finite automaton</u>, denoted by "AFA" ("NFA", "DFA"). A three-way AFA (NFA, DFA) is denoted by "TAFA" ("TNFA", "TDFA"). For any positive integer k, a <u>two-dimensional alternating (nondeterministic, deterministic) k-marker automaton</u>, denoted by "AMA(k)" ("NMA(k)", "DMA(k)"), is an AFA (NFA, DFA) which can use k markers on the input tape. By "AFA^s" we denote an AFA whose input tapes are restricted to square ones. NFA^s, DFA^s, etc., have the same meaning. Define

$\mathcal{L}[ATM(L(m,n))]=\{T \mid T=T(M) \text{ for some } ATM(L(m,n))\ M\}$, and

$\mathcal{L}[ATM^s(L(m))]=\{T \mid T=T(M) \text{ for some } ATM^s(L(m))\ M\}$.

$\mathcal{L}[NTM(L(m,n))]$, $\mathcal{L}[NTM^s(L(m))]$, $\mathcal{L}[AFA]$, $\mathcal{L}[AFA^s]$, etc., have the same meaning.

The following concepts are used in the subsequent discussions.

<u>Definition 2.2</u>. A function $L(m):N\rightarrow R$ ($L(m,n):N^2\rightarrow R$) is called <u>two-dimensionally space constructible</u> if there is a DTM^s (DTM) M such that (i) for each $m\geq 1$ ($m,n\geq 1$) and for each input tape x with $\mathcal{Q}_1(x)=\mathcal{Q}_2(x)=m$ ($\mathcal{Q}_1(x)=m$ and $\mathcal{Q}_2(x)=n$), M uses at most $\lceil L(m)\rceil$ ($\lceil L(m,n)\rceil$) cells of the storage tape, (ii) for each $m\geq 1$ ($m,n\geq 1$), there exists some input tape x with $\mathcal{Q}_1(x)=\mathcal{Q}_2(x)=m$ ($\mathcal{Q}_1(x)=m$ and $\mathcal{Q}_2(x)=n$) on which M halts after its storage head has marked off exactly $\lceil L(m)\rceil$ ($\lceil L(m,n)\rceil$) cells of the storage tape, and (iii) for each $m\geq 1$ ($m,n\geq 1$), when given any input tape x with $\mathcal{Q}_1(x)=$

$Q_2(x)=n$ ($Q_1(x)=n$ and $Q_2(x)=n$), M never halts without marking off exactly $\lceil L(n) \rceil$ ($\lceil L(n,n) \rceil$) cells of the storage tape.

Definition 2.3. A function $L(n):N\to R$ ($L(n,n):N^2\to R$) is called two-dimensionally fully space constructible if there exists a DTM^s (DTM) M which, for each $n\geq 1$ ($n,n\geq 1$) and for each input tape x with $Q_1(x)=Q_2(x)=n$ ($Q_1(x)=n$ and $Q_2(x)=n$), makes use of exactly $\lceil L(n) \rceil$ ($\lceil L(n,n) \rceil$) cells of the storage tape and halts.

Notation 2.1. Let f(n) and g(n) be any functions with one variable n. We write $f(n)<<g(n)$ when $\lim_{n\to\infty} f(n)/g(n)=0$.

2.2. A Difference among Alternating, Nondeterministic, and Deterministic Machines

This section states a difference among the accepting powers of alternating, nondeterministic, and deterministic machines. For the one-dimensional case, it is well known [11,24,69] that the following theorem holds.

Theorem 2.1. For any function $L(n)<<\log\log n$, L(n) space-bounded two-way alternating, nondeterministic, and deterministic Turing machines are all equivalent to one-way deterministic finite automata in accepting power.

We first show that a different situation occurs for the two-dimensional case. Let $T_1=\{x\in\{0,1\}^{(2)} \mid \exists n\geq 1[\ Q_1(x)=Q_2(x)=n \ \&\ \exists i(1\leq i\leq n-1)[x[(i,1),(i,n)]=x[(n,1),(n,n)]]]\}$ and $T_2=\{x\in\{0,1\}^{(2)} \mid \exists n\geq 0[\ Q_1(x)=Q_2(x)=2n+1 \ \&\ x(n+1,n+1)=1$(i.e., the center symbol of x is 1)$]\}$. It is shown in [58,59] that $T_1\in\mathcal{L}[TAFA^s]-\mathcal{L}[NTM^s(L(n))]$ and $T_2\in\mathcal{L}[TNFA^s]-\mathcal{L}[DTM^s(L(n))]$ for any function $L(n)<<\log n$. Thus we have

Theorem 2.2. For any function $L(n)<<\log n$, (1) $\mathcal{L}[DTM^s(L(n))]\subsetneq\mathcal{L}[NTM^s(L(n))]\subsetneq\mathcal{L}[ATM^s(L(n))]$, and (2) $\mathcal{L}[TDTM^s(L(n))]\subsetneq\mathcal{L}[TNTM^s(L(n))]\subsetneq\mathcal{L}[TATM^s(L(n))]$.

Corollary 2.1 [3,58,59,89]. $\mathcal{L}[DFA^s]\subsetneq\mathcal{L}[NFA^s]\subsetneq\mathcal{L}[AFA^s]$ and $\mathcal{L}[TDFA^s]\subsetneq\mathcal{L}[TNFA^s]\subsetneq\mathcal{L}[TAFA^s]$.

For the three-way case, we can show that the following stronger results hold.

Theorem 2.3. (1) $\mathcal{L}[TDTM^s(L(n))]\subsetneq\mathcal{L}[TNTM^s(L(n))]\subsetneq\mathcal{L}[TATM^s(L(n))]$ for any function $L(n)<<n^2$, (2) $\mathcal{L}[TDTM(L(n,n))]\subsetneq\mathcal{L}[TNTM(L(n,n))]\subsetneq\mathcal{L}[TATM(L(n,n))]$ for each $L(n,n)\in\{f(n)\times g(n),\ f(n)+g(n)\}$, where $f(n):N\to R$ is a function such that $f(n)<<n$, and $g(n):N\to R$ is a monotone nondecreasing function which is fully space constructible [25], and (3) $\mathcal{L}[TDTM(L(n,n))]\subsetneq\mathcal{L}[TNTM(L(n,n))]\subsetneq\mathcal{L}[TATM(L(n,n))]$ for each $L(n,n)\in\{f(n)\times g(n),\ f(n)+g(n)\}$, where $f(n):N\to R$ is a function, and $g(n):N\to R$ is a function such that $g(n)<<n$.

Proof. (1): See [44,58].

(2): In [44], it is shown that $\mathcal{L}[TDTM(L(n,n))]\subsetneq\mathcal{L}[TNTM(L(n,n))]$. Below, we show that $\mathcal{L}[TNTM(L(n,n))]\subsetneq\mathcal{L}[TATM(L(n,n))]$. Let $T[g]=\{x\in\{0,1\}^{(2)} \mid \exists n\geq 1[\ Q_1(x)=2\times 2^{\lceil g(n) \rceil} \ \&\ Q_2(x)=n \ \&$ (the top and bottom halves of x are the same)$]\}$. It is easy to show that $T[g]\in\mathcal{L}[TATM(g(n))]$. The claim follows from this and from the fact [44] that $T[g]\notin\mathcal{L}[TNTM(L(n,n))]$ for each $L(n,n)\in\{f(n)\times g(n),\ f(n)+g(n)\}$.

(3): In [44], it is shown that $\mathcal{L}[TDTM(L(n,n))]\subsetneq\mathcal{L}[TNTM(L(n,n))]$. Below, we show that $\mathcal{L}[TNTM(L(n,n))]\subsetneq\mathcal{L}[TATM(L(n,n))]$. Let $T_3=\{x\in\{0,1\}^{(2)} \mid Q_1(x)=2$ & (the first and second rows of x are the same)$\}$. It is easy to show that $T_3\in\mathcal{L}[TAFA]$. The claim follows from this and from the fact [44] that $T_3\notin\mathcal{L}[TNTM(L(n,n))]$ for each $L(n,n)\in\{f(n)\times g(n),\ f(n)+g(n)\}$.

For four-way Turing machines on nonsquare tapes, we have

Theorem 2.4. (1) $\mathcal{L}[NTM(L(n,n))]\subsetneq\mathcal{L}[ATM(L(n,n))]$ for each $L(n,n)\in\{f(n)\times g(n),\ f(n)+g(n)\}$, where $f(n):N\to R$ is a function such that $f(n)<<\log n$, and $g(n):N\to R$ is a monotone nondecreasing function which is fully space constructible. (2) $\mathcal{L}[NTM(L(n,n))]\subsetneq\mathcal{L}[ATM(L(n,n))]$ for each $L(n,n)\in\{f(n)\times g(n),\ f(n)+g(n)\}$, where $f(n):N\to R$ is a monotone nondecreasing function which is fully space con-

structible, and $g(n):N\rightarrow R$ is a function such that $g(n)<<\log n$.

Proof. We only prove (1), because the proof of (2) is similar. Let $x\in\{0,1\}^{(2)}$ and $\Omega_2(x)=n$ $(n\geq 1)$. When $\Omega_1(x)$ is divided by $2\lceil g(n)\rceil$, we call

$$x[(j-1)2\lceil g(n)\rceil+1,1),(j2\lceil g(n)\rceil,n)]$$

the j-th g(n)-block of x for each j $(1\leq j\leq \Omega_1(x)/2\lceil g(n)\rceil)$. We say that x has exactly k g(n)-blocks if $\Omega_2(x)=n$ and $\Omega_1(x)=k2\lceil g(n)\rceil$ for some positive integer $k\geq 1$. Let $T(g)=\{x\in\{0,1\}^{(2)} \mid (\exists n\geq 1)(\exists k\geq 2)[(x$ has exactly k g(n)-blocks) & $\exists j(2\leq j\leq k)[$the first and j-th g(n)-blocks of x are identical]]$\}$. It is easy to show that $T(g)\in\mathcal{L}[ATM(g(n))]$. On the other hand, we can show, by using the same technique as in the proof of Lemma 3.3 in [45], that $T(g)\notin\mathcal{L}[NTM(L(m,n))]$ for each $L(m,n)\in\{f(m)\times g(n),\ f(m)+g(n)\}$. Thus (1) follows.

It is well known [3] that one-dimensional 1-marker automata are equivalent to one-dimensional finite automata. For the two-dimensional case, a different situation occurs. Let T_1 be the set described above. We can show that $T_1\in\mathcal{L}[DMA(1)]-\mathcal{L}[NFA]$. Let $T_4=\{x\in\{0,1\}^{(2)} \mid \exists m\geq 1[\ \Omega_1(x)=2m$ & $\Omega_2(x)=m$ & (the top and bottom halves of x are the same)]$\}$. It is shown in [29,113] that $T_4\in\mathcal{L}$ $[NMA(1)]-\mathcal{L}[DMA(1)]$. Thus we have

Theorem 2.5. (1) There exists a set in $\mathcal{L}[DMA(1)]$, but not in $\mathcal{L}[NFA]$, and (2) $\mathcal{L}[DMA(1)]\subsetneq\mathcal{L}$ $[NMA(1)]$.

Savitch [91] showed that for any fully space constructible function $L(n)\geq\log n$, L(n) space-bounded one-dimensional nondeterministic Turing machines can be simulated by $L^2(n)$ space-bounded one-dimensional deterministic Turing machines. By using the same technique as in [91], we can show that a similar result also holds for the two-dimensional case.

Theorem 2.6. For any two-dimensionally fully space constructible function $L(m)\geq\log m$ $(L(m,n)\geq\log m+\log n)$, $\mathcal{L}[NTM^s(L(m))]\subseteq\mathcal{L}[DTM^s(L^2(m))]$ $(\mathcal{L}[NTM(L(m,n))]\subseteq\mathcal{L}[DTM(L^2(m,n))])$.

Open problems: (1) For any two-dimensionally fully space constructible function $L(m)\geq\log m$ $(L(m,n)\geq\log m+\log n)$, $\mathcal{L}[DTM^s(L(m))]\subsetneq\mathcal{L}[NTM^s(L(m))]\subsetneq\mathcal{L}[ATM^s(L(m))]$ ($\mathcal{L}[DTM(L(m,n))]\subsetneq\mathcal{L}$ $[NTM(L(m,n))]\subsetneq\mathcal{L}[ATM(L(m,n))]$) ? (2) Let f(m) and g(n) be the functions described in Theorem 2.4(1) or Theorem 2.4(2). Then $\mathcal{L}[DTM(L(m,n))]\subsetneq\mathcal{L}[NTM(L(m,n))]$ for each $L(m,n)\in\{f(m)\times g(n),$ $f(m)+g(n)\}$? (3) Is there a set in $\mathcal{L}[NFA]$, but not in $\mathcal{L}[DMA(1)]$? (4) For any $k\geq 1$, $\mathcal{L}[DMA(k)]\subsetneq$ $\mathcal{L}[NMA(k)]\subsetneq\mathcal{L}[AMA(k)]$?

2.3. Three-way versus Four-way

This section states a relationship between the accepting powers of three-way machines and four-way machines.

As shown in Theorem 2.1, for the one-dimensional case, L(n) space-bounded one-way and two-way Turing machines are equivalent for any $L(n)<<\log\log n$. We shall below show that a different situation occurs for the two-dimensional case. Let $T_5=\{x\in\{0,1\}^{(2)} \mid \exists m\geq 1\ [\ \Omega_1(x)=\Omega_2(x)=2m$ & $(x[(1,1),(1,m)]$ is the reversal of $x[(1,m+1),(1,2m)])]\}$. It is shown in [64] that $T_5\in\mathcal{L}[DFA^s]-\mathcal{L}$ $[TATM^s(L(m))]$ for any function $L(m)<<\log m$. On the other hand, as stated in Section 2.2, $T_1\in\mathcal{L}$ $[TAFA^s]-\mathcal{L}[NTM^s(L(m))]$ for any $L(m)<<\log m$. From these facts, for example, we have

Theorem 2.7. For any function $L(m)<<\log m$, (1) $\mathcal{L}[TXTM^s(L(m))]\subsetneq\mathcal{L}[XTM^s(L(m))]$ for each $X\in$ $\{D,N,A\}$, (2) $\mathcal{L}[DTM^s(L(m))]$ is incomparable with $\mathcal{L}[TNTM^s(L(m))]$ and $\mathcal{L}[TATM^s(L(m))]$, and (3) $\mathcal{L}$ $[NTM^s(L(m))]$ is incomparable with $\mathcal{L}[TATM^s(L(m))]$.

Remark 2.1. It is shown in [44] that Theorem 2.7(1) can be strengthened as follows: " $\mathcal{L}$ $[TXTM^s(L(m))]\subsetneq\mathcal{L}[XTM^s(L(m))]$ for each $X\in\{D,N\}$ and each function $L(m)<<m^2$." It is obvious that $\mathcal{L}[TXTM^s(L(m))]=\mathcal{L}[XTM^s(L(m))]$ for each $L(m)\geq m^2$.

Remark 2.2. By using the same technique as in the proof of the fact [74] that L(n) space-bounded

one-way and two-way alternating Turing machines are equivalent for any $L(n)\geq\log n$, we can show that $\mathcal{L}[TATM^s(L(n))]=\mathcal{L}[ATM^s(L(n))]$ for any function $L(n)\geq\log n$.

For nonsquare tapes, we have

Theorem 2.8. (1) $\mathcal{L}[TXTM(L(m,n))]\subsetneq\mathcal{L}[XTM(L(m,n))]$ for each $X\in\{D,N\}$ and each $L(m,n)\in\{f(m)\times g(n),\ f(m)+g(n)\}$, where $f(m)$ and $g(n)$ are the functions described in Theorem 2.3(2) or Theorem 2.3(3), (2) $\mathcal{L}[TATM(L(m,n))]\subsetneq\mathcal{L}[ATM(L(m,n))]$ for each $L(m,n)\in\{f(m)\times g(n),\ f(m)+g(n)\}$, where $f(m):N\to R$ is a function such that $f(m)<<\log m$, and $g(n):N\to R$ is a monotone nondecreasing function which is fully space constructible, and (3) $\mathcal{L}[TATM(L(m,n))]=\mathcal{L}[ATM(L(m,n))]$ for any function $L(m,n)\geq\log m$.

Proof. See [44] for (1). We leave the proof of (3) to the reader. We below show that (2) holds. Let $T(g)$ be the set described in the proof of Theorem 2.4 (1). As stated in the proof of Theorem 2.4(1), $T(g)\in\mathcal{L}[ATM(g(n))]$. On the other hand, we can show, by using the same technique as in the proof of Lemma 4.2 in [64], that $T(g)\notin\mathcal{L}[TATM(L(m,n))]$ for each $L(m,n)\in\{f(m)\times g(n),\ f(m)+g(n)\}$. Thus it follows that (2) holds.

It is natural to ask how much space is required for three-way machines to simulate four-way machines. The following two theorems answer this question.

Theorem 2.9. (1) n log n (n^2) space is necessary and sufficient for TDTM's to simulate DFA's (NFA's) (see [48,83]). (2) n space is necessary and sufficient for TNTM's to simulate DFA's and NFA's (see [57]). (3) $2^{\Theta(n\log n)}$ ($2^{\Theta(n^2)}$) space is necessary and sufficient for TDTM's to simulate DMA(1)'s (NMA(1)'s) (see [67]). (4) n log n (n^2) space is necessary and sufficient for TNTM's to simulate DMA(1)'s (NMA(1)'s) (see [67]). (In this theorem, note that n denotes the number of columns of tapes.)

Open problems: (1) $\mathcal{L}[AFA]\subseteq\mathcal{L}[TNTM(n)]$? (2) $\mathcal{L}[AMA(1)]\subseteq\mathcal{L}[TNTM(2^{O(n)})]$?

2.4. Two-Dimensionally Space Constructible Functions and Space Complexity Results

This section concerns two-dimensionally space constructible functions and space complexity hierarchy. We state these subjects only for square tapes. (See [78,80,82] for the case of nonsquare tapes.) It is well known [24] that in the one-dimensional case, there exists no space constructible function which grows more slowly than the order of loglog n, thus no space hierarchy of language acceptability exists below space complexity loglog n. Below, we state that a different situation occurs for the two-dimensional case.

We consider the following three functions:

(i) $\log^{(1)}n=\begin{cases}0 & (n=0)\\ \lceil\log_2 n\rceil & (n\geq 1)\end{cases}$

$\log^{(k+1)}n=\log^{(1)}(\log^{(k)}n)$

(ii) $\exp^*0=1$, $\exp^*(n+1)=2^{\exp^* n}$

(iii) $\log^* n=\min\{x\mid \exp^* x\geq n\}$.

The following theorem demonstrates that there exist two-dimensionally space constructible functions which grow more slowly than the order of loglog n.

Theorem 2.10 [78,82]. The functions $\log^k n$ (k: any natural number) and $\log^* n$ are two-dimensionally space constructible.

More generally, we have

Theorem 2.11 [78,82]. Let $f(n):N\to N$ be any monotone nondecreasing total recursive function such that $\lim_{n\to\infty}f(n)=\infty$. Then, there exists a two-dimensionally space constructible and monotone nondecreasing function $L(n)$ such that (i) $L(n)<f(n)$ and (ii) $\lim_{n\to\infty}L(n)=\infty$.

It is shown in [105] that there exists no fully space constructible function which grows more

slowly than the order of log n. It is unknown whether or not there exists a two-dimensionally fully space constructible function which grows more slowly than the order of log n.

For the one-dimensional case, the following three important theorems concerning space complexity hierarchy of Turing machines are known. (By $\mathcal{L}[1NTM(L(n))]$ ($\mathcal{L}[1DTM(L(n))]$) we denote the class of languages accepted by L(n) space-bounded one-dimensional nondeterministic (deterministic) Turing machines [25].)

Theorem 2.12 [102]. Let L(n) be a space function. For any constant $c>0$ and each $X\in\{D,N\}$, $\mathcal{L}[1XTM(L(n))]=\mathcal{L}[1XTM(c\cdot L(n))]$.

Theorem 2.13 [102]. Let $L_1(n)$ and $L_2(n)$ be any space constructible functions such that $\lim_{i\to\infty} L_1(n_i)/L_2(n_i)=0$ and $L_2(n_i)/\log n_i>k$ $(i=1,2,\cdots)$ for some increasing sequence of natural numbers $\{n_i\}$ and for some constant $k>0$. Then there exists a language in $\mathcal{L}[1DTM(L_2(n))]$, but not in $\mathcal{L}[1DTM(L_1(n))]$.

Theorem 2.14 [24]. Let $L_1(n)$ and $L_2(n)$ be space constructible functions such that $\lim_{i\to\infty} L_1(n_i)/L_2(n_i)=0$ and $L_2(n_i)/\log n_i<1/2$ for some increasing sequence of natural numbers $\{n_i\}$. Then there exists a language in $\mathcal{L}[1DTM(L_2(n))]$, but not in $\mathcal{L}[1DTM(L_1(n))]$.

By using the ideas similar to those of the proofs of Theorems 2.12 and Theorem 2.13, we can prove the following two-dimensional analogues to these theorems.

Theorem 2.15. Let L(m) be a space function. For any constant $c>0$ and each $X\in\{D,N,A\}$,

$$\mathcal{L}[XTM^s(L(m))]=\mathcal{L}[XTM^s(cL(m))].$$

Theorem 2.16 [78,80]. Let $L_2(m)$ be a two-dimensionally space constructible function. Suppose that $\lim_{i\to\infty} L_1(m_i)/L_2(m_i)=0$ and $L_2(m_i)>k\cdot\log m_i$ $(i=1,2,\cdots)$ for some increasing sequence of natural numbers $\{m_i\}$ and for some constant $k>0$. Then there exists a set in $\mathcal{L}[DTM^s(L_2(m))]$ but not in $\mathcal{L}[DTM^s(L_1(m))]$.

Recently, It is shown in [28,103] that for each space constructible function $L(n)\geq\log n$, $\mathcal{L}[1NTM(L(n))]$ is closed under complementation. This result can be extended to the two-dimensional case. By using these facts, we can extend Theorem 2.13 and Theorem 2.16 to the nondeterministic case [21].

The following theorem, which is a two-dimensional analogue to Theorem 2.14, cannot be proved by the same idea as in the proof of Theorem 2.14.

Theorem 2.17 [78,80]. Let $L_2(m)$ be a two-dimensionally space constructible function. Suppose that $\lim_{i\to\infty} L_1(m_i)/L_2(m_i)=0$, $\lim_{i\to\infty} L_2(m_i)=\infty$, and $L_2(m_i)<k\cdot\log m_i$ $(i=1,2,\cdots)$ for some increasing sequence of natural numbers $\{m_i\}$ and for some constant $k>0$. Then there exists a set in $\mathcal{L}[DTM^s(L_2(m))]$, but not in $\mathcal{L}[DTM^s(L_1(m))]$.

The following theorem, which is a nondeterministic version of Theorem 2.17, is proved in [60].

Theorem 2.18 [60]. Let $L_2(m)$ be a two-dimensionally space constructible function such that $L_2(m)\leq\log m$. Suppose that $\lim_{m\to\infty} L_1(m)/L_2(m)=0$. Then there exists a set in $\mathcal{L}[NTM^s(L_2(m))]$ (in fact, in $\mathcal{L}[DTM^s(L_2(m))]$) but not in $\mathcal{L}[NTM^s(L_1(m))]$.

From Theorem 2.10 and Theorem 2.18, we have the following corollary, which implies that in the two-dimensional case, there is an infinite hierarchy of acceptabilities even for space complexity classes below loglog n.

Corollary 2.2. For any constant $c>0$, each $k\in N$, and each $X\in\{D,N\}$,

$$\mathcal{L}[XFA^s]=\mathcal{L}[XTM^s(c)]\subsetneq\cdots\subsetneq\mathcal{L}[XTM^s(\log^{(k+1)}m)]\subsetneq\mathcal{L}[XTM^s(\log^{(k)}m)]\cdots$$

Open problem: Do results analogous to Theorems 2.16 and 2.17 hold for ATM^s ?

2.5 Closure properties

This section presents only closure properties of the classes of sets accepted by several types of

two-dimensional finite automata. (See [41,44,45,48,106] for closure properties of the classes of sets accepted by space-bounded two-dimensional Turing machines.) It is well known [25] that the class of sets accepted by one-dimensional finite automata is closed under many operations , including Boolean operations. We below demonstrate that a different situation occurs for two-dimensional finite automata. We first define several operations over two-dimensional tapes.

Definition 2.4. Let

$$x=\begin{matrix} a_{11}\ldots a_{1n} \\ \vdots \\ a_{m1}\ldots a_{mn} \end{matrix}, \text{ and } y=\begin{matrix} b_{11}\ldots b_{1n'} \\ \vdots \\ b_{m'1}\ldots b_{m'n'} \end{matrix}$$

Then the rotation x^R of x and the row reflection x^{RR} of x are given by Fig.2 and Fig.3, respectively. A row cyclic shift of x is any two-dimensional tape of the form of Fig.4 for some $1\leq k\leq m$ (not that for k=m this is x itself), and a column cyclic shift of x is any two-dimensional tape of the form of Fig.5 for some $1\leq k\leq n$ (not that for k=n this is x itself). The row catenation $x\ominus y$ is defined only when n=n' and is given by Fig.6, and the column catenation $x\oplus y$ is defined only when m=m' and is given by Fig.7.

$$\begin{matrix} a_{m1}\ldots a_{11} \\ \vdots \\ a_{mn}\ldots a_{1n} \end{matrix}$$

Fig.2

$$\begin{matrix} a_{m1}\ldots a_{mn} \\ \vdots \\ a_{11}\ldots a_{1n} \end{matrix}$$

Fig.3

$$\begin{matrix} a_{k+1,1}\ldots a_{k+1,n} \\ \vdots \\ a_{m1}\ldots a_{mn} \\ a_{11}\ldots a_{1n} \\ \vdots \\ a_{k1}\ldots a_{kn} \end{matrix}$$

Fig.4

$$\begin{matrix} a_{11}\ldots a_{1n} \\ \vdots \\ a_{m1}\ldots a_{mn} \\ b_{11}\ldots b_{1n} \\ \vdots \\ b_{m'1}\ldots b_{m'n} \end{matrix}$$

Fig.6

$$\begin{matrix} a_{1,k+1}\ldots a_{1n}\ a_{11}\ldots a_{1k} \\ \vdots \\ a_{m,k+1}\ldots a_{mn}\ a_{m1}\ldots a_{mk} \end{matrix}$$

Fig 5.

$$\begin{matrix} a_{11}\ldots a_{1n}\ b_{11}\ldots b_{1n'} \\ \vdots \\ a_{m1}\ldots a_{mn}\ b_{m1}\ldots b_{mn'} \end{matrix}$$

Fig.7

Definition 2.5. Let S and S' be two sets of two-dimensional tapes. Then

$S^R=\{x^R \mid x\in S\}$ (rotation of S),

$S^{RR}=\{x^{RR} \mid x\in S\}$ (row reflection of S),

$S^{RC}=\{y \mid y$ is a row cyclic shift of some $x\in S\}$ (row cyclic closure of S),

$S^{CC}=\{y \mid y$ is a column cyclic shift of some $x\in S\}$ (column cyclic closure of S).

$S\ominus S'=\{x\ominus y \mid x$ in S, y in $S'\}$ (row catenation),

$S\oplus S'=\{x\oplus y \mid x$ in S, y in $S'\}$ (column catenation),

$S^+=\bigcup_{i\geq 1} S^i$ (row closure),

$S^*=\bigcup_{i\geq 1} S^i$ (column closure),

where $S^1=S$, $S^2=S\ominus S,\ldots$, $S^{i+1}=S^i\ominus S$, and $S^1=S$, $S^2=S\oplus S,\ldots$, $S^{i+1}=S^i\oplus S$.

For three-way finite automata, we have

Theorem 2.19. (1) $\mathcal{L}$[TDFA] is not closed under union, intersection, rotation, row reflection, row and column cyclic closures, row and column catenations, or row and column closures [44,45,48,56,106]. (2) $\mathcal{L}$[TNFA] is closed under union, row catenation, and row closure, but not closed under intersection, complementation, rotation, row and column cyclic closures, column catenation, or column closure [44,45,56,106]. (3) $\mathcal{L}$[TAFA] is closed under union and intersection, but not closed under rotation, row reflection, row and column cyclic closures, row and column catenations, or row and column closures [64,68].

Open problems: (1) Are $\mathcal{L}$[TDFA] and $\mathcal{L}$[TAFA] closed under complementation ? (2) Is $\mathcal{L}$[TNFA] closed under row reflection ?

For four-way finite automata, we have

Theorem 2.20. (1)$\mathcal{L}$[DFA] is closed under Boolean operations, rotation and row reflection, but not closed under row and column cyclic closures, row and column catenations, or row and column closures [41,42,51]. (2)$\mathcal{L}$[NFA] is closed under union, intersection, rotation, and row reflection, but not closed under row and column cyclic closures, row and column catenations, or row and column closures [41,51,52]. (3) $\mathcal{L}$[AFA] is closed under union and intersection, rotation, and row reflection.

Remark 2.3. That $\mathcal{L}$[DFA] is closed under Boolean operations can be proved by using the technique in [96].

Open problems: (1) Is $\mathcal{L}$[NFA] closed under complementation ? (2) Is $\mathcal{L}$[AFA] closed under complementation, row and column cyclic closures, row and column catenations, and row and column closures ?

2.6. Decision Problems

This section concerns decision problems of two-dimensional finite automata. It is well known [25] that many decision problems of one-dimensional finite automata are decidable. As suggested by the following theorem, most of decision problems of four-way two-dimensional finite automata are undecidable.

Theorem 2.21 [3,111]. The emptiness and universe problems for DFA's are undecidable even for a one-letter alphabet.

We below state some decision problems of three-way finite automata. For each $X \in \{D,N,A\}$, let TXFA(O) denote a TXFA which operates on two-dimensional tapes over a one-letter alphabet. The following two theorems are all that have been obtained for three-way finite automata by now.

Theorem 2.22 [49]. (1) The emptiness and universe problems for TDFA(O)'s are decidable. (2) The emptiness problem for TNFA(O)'s is decidable. (3) The universe, inclusion, and equivalence problems for TNFA's are undecidable.

Theorem 2.23 [70]. (1) The disjointness, inclusion, and equivalence problems for TDFA(O)'s are decidable. (2) The disjointness and inclusion problems for TDFA's are undecidable.

Open problems: (1) Are the emptiness, universe, and equivalence problems for TDFA's decidable ? (2) Are the universe, inclusion, and equivalence problems for TNFA(O)'s and TAFA(O)'s decidable ? (3) Is the emptiness problem for TAFA(O)'s decidable ?

2.7. Recognizability of Connected Pictures

Let T_c be the set of all two-dimensional connected pictures [53,89]. It is interesting to investigate how much space is required for two-dimensional Turing machines to accept T_c. For this problem, we have

Theorem 2.24. (1) n space is necessary and sufficient for TDTM's and TNTM's to accept T_c (see [116]). (2) $T_c \in \mathcal{L}$[AFA] (see [58]). (3) $T_c \in \mathcal{L}$[DMA(1)] (see [3,89]). (4) $T_c^s \notin \mathcal{L}[TATM^s(L(m))]$ for any $L(m) << \log m$, where T_c^s denotes the set of all the square connected pictures (see [64]).

Open problem: $T_c \in \mathcal{L}$[DFA] or $T_c \in \mathcal{L}$[NFA] ?

2.8. Other Topics

In this section, we list up other topics and related references about sequential automata on a two-dimensional tape.

(1) Maze (or labyrinth) search problems: see [1,4,5,7,8,9,10,22,75,104].

(2) Characterizations of one-dimensional languages by two-dimensional automata: see [20,29,32,33].

(3) A relationship between two-dimensional automata and two-dimensional array grammars: see [19,30,73,76,79,84,89,99,115].

(4) Properties of special types of two-dimensional Turing machines (two-dimensional pushdown automata, stack automata, multi-counter automata, multihead automata, and marker automata): see [3,27,46,47,55,56,78,81,89,94,95,113].

(5) Parallel, time, space, and reversal complexities of two-dimensional alternating multihead Turing machines: see [26,50,58,59].

(6) Properties of two-dimensional finite automata over a one-letter alphabet: see [36,40,70].

(7) Properties of two-dimensional automata on a nonrectangular tape: see [77,88,89].

(8) A relationship between two-dimensional alternating finite automata and cellular types of two-dimensional automata: see [62,63,65,66].

The most interesting problem in the future is to investigate time complexity hierarchy of two-dimensional Turing machines.

Two-dimensional (or array) grammars are not discussed here. For this subject, see the excellent book of Rosenfeld [89] and the excellent surveys of Siromoney [97,98].

3. Cellular Types of Two-Dimensional Automata

Many authors investigated language acceptability of one-dimensional cellular automata (for example, see [6,12,14,101,114]). On the other hand, cellular automata on a two-dimensional tape are being investigated not only in the viewpoint of formal language theory but also in the viewpoint of pattern recognition. Cellular automata on a two-dimensional tape can be classified into three types.

The first type, called a two-dimensional cellular automaton (CA for short), is investigated in [2,13,17,29,31,34,35,37,39,53,61-63,65,71,72,87,89,100,112]. CA's make use of two-dimensional cellular arrays. It is shown, for example, that (1) the set T_c of all two-dimensional connected pictures can be accepted by deterministic CA's in linear time [2], (2) the majority problem can be solved by deterministic CA's in linear time, and thus the set of all the two-dimensional tapes over {0,1} with positive Euler number can be accepted by deterministic CA's in linear time [100], (3) the two-dimensional packing problem can be solved by deterministic CA's in linear time [71], (4) NFA's can be simulated by deterministic CA's in linear time [72], and (5) AFA's can be simulated by deterministic CA's in constant state change [62]. (The notion of state change complexity was first introduced in [114]. Many properties of two-dimensional on-line tessellation acceptors (OTA's for short) introduced in [29,35] are investigated in [29,31,34,35,37,39,53,65,112]. The OTA is a restricted type of CA in which cells do not make transitions at every time step; rather, a transition 'wave' passes once diagonally across the array. It is shown, for example, that (1) nondeterministic OTA's are more powerful than NFA's, and deterministic OTA's are incomparable (in accepting power) with NFA's and DFA's [29,35], (2) the set T_c described above cannot be accepted by deterministic OTA's [53], and (3) deterministic OTA's can be used as two-dimensional pattern matching machines [112]. In [17], a generalization of CA's in which each cell is a space-bounded Turing machine rather than a finite automaton, is introduced. Fast algorithms are given for performing various basic image processing tasks by such automata.

The second type of cellular automata on a two-dimensional tape is investigated in [15,30-33,37,38,57,66,78,89,90,92,93,99,107-110].Two typical models of this type are parallel/sequential

array automata (PSA's) [90] and one-dimensional bounded cellular acceptors (BCA's) [92,107-110]. The PSA makes use of one-dimensional (e.g., horizontal) cellular array which can move, as a unit, in the vertical direction, and accepts a tape if the leftmost cell (i.e., the cell which reads the first column of the tape) enters an accepting state in some time. The BCA is a restricted type of one-way PSA in which the cellular array moves downwards each time step, and the BCA accepts a tape if the state configuration of the cellular array just after it has completely scanned the tape is an element of the specified regular set (called the accepting configuration set). It is shown, for example, that (1) nondeterministic one-way PSA's are more powerful than deterministic ones, two-way PSA's are more powerful than one-way PSA's, and T_c is accepted by deterministic one-way PSA's [90], (2) deterministic one-way PSA's are incomparable with NFA's and DFA's [15,48], (3) one-way PSA's are more powerful than OTA's [35], (4) nondeterministic BCA's are equivalent to nondeterministic OTA's, and deterministic BCA's are incomparable with deterministic OTA's and DFA's [31,57]. See [30,99] for a relationship between PSA's and two-dimensional grammars, and see [37,38] for closure properties of PSA's. An extension of BCA's in which the accepting configuration set is a context-free language, context-sensitive language, or phrase structure language, is introduced in [107-110].

The third type, called a pyramid cellular acceptor (PCA), is investigated in [16,18,43,54,85,86,89]. The PCA is a pyramid stack of two-dimensional cellular arrays, where the bottom array has size 2^n by 2^n, the next lowest 2^{n-1} by 2^{n-1}, and so forth, the (n+1)st layer consisting of a single cell, called the root. Each cell has nine neighbors -- four son cells in a 2-by-2 block in the level below, four brother cells in the current level, and one father cell in the level above. The transition function of each cell maps 10-tuples of states into states -- or sets of states, in the nondeterministic case. An input tape is stored as initial states of the bottom array; the upper-level cells are initialized to a quiescent state. The root is the accepting state. A bottom-up pyramid cellular acceptor (UPCA) is a PCA in which the next state of a cell depends only on the current states of that cell and its four sons. It is shown, for example, that (1) both nondeterministic PCA's and nondeterministic UPCA's are equivalent to nondeterministic CA'S [16,85,89], (2) nondeterministic UPCA's are more powerful than deterministic UPCA's [85,89], (3) nondeterministic UPCA's can simulate nondeterministic OTA's, thus NFA's in O(diameter) time [54,86], and (4) O(diameter $\times$ log diameter) time (O((diameter)^2) time) is necessary for deterministic UPCA's to simulate DFA's (NFA's) [54]. See the excellent book [89] of Rosenfeld for image processing task by PCA's and UPCA's.

Open problems:

(1) Can AFA's be simulated by deterministic CA's in linear time ?

(2) Can AFA's be simulated by nondeterministic OTA's ?

(3) Are deterministic CA's equivalent to nondeterministic CA's ?

(4) Is T_c accepted by nondeterministic OTA's or deterministic UPCA's ?

(5) Is T_c accepted by nondeterministic UPCA's in diameter time ?

(6) Can DFA's, NFA's, or AFA's be simulated by deterministic UPCA's ?

4. Conclusions

In this paper, we surveyed several aspects of two-dimensional automata theory. We believe that there are many problems about two-dimensional automata to solve in the future. We hope that this survey will activate the investigation of two-dimensional automata theory.

References

[1] H.Antelmann, L.Budach and H.A.Rollik, On universal traps, EIK 15, 3 (1979),123-131.
[2] T.Beyer, Recognition of topological invariants by iterative arrays, Ph.D.Thesis, MIT, 1970.
[3] M.Blum and C.Hewitt, Automata on a two-dimensional tape, IEEE Symposium on Switching and Automata Theory, 1967, 155-160.
[4] M.Blum and D.Kozen, On the power of the compass, Proc. of the 19th Annual Symp. on Foundations of Computer Science, 1978,
[5] M.Blum and W.J.Sakoda, On the capability of finite automata in 2 and 3 dimensional space, Proc. of the 18th Annual Symp. on Foundations of Computer Science, 1977, 147-161.
[6] W.Bucher and K.Culik II, On real time and linear time cellular automata, Research Report 115, Institute fur Informationsverarbeitung, Technical University of Graz, 1983.
[7] L.Budach, Environments, labyrinths and Automata, Lecture Notes in Computer Science 56, Springer Verlag, 1977.
[8] __, Automata and Labyrinths, Math. Nachr. 86 (1978), 195-282.
[9] L.Budach and C.Meinel, Environments and automata I, EIK 18, 1/2 (1982), 13-40.
[10] __, Environments and automata II, EIK 18, 3 (1982), 115-139.
[11] A.K.Chandra, D.C.Kozen and L.J.Stockmeyer, Alternation, J.Assoc.Comput.Mach. 28, 1 (1981), 114-133.
[12] C.Choffrut and K.Culik II, On real-time cellular automata and trellis automata, Research Report F114, Institute fur Informationsverarbeitung, Technical University of Graz, 1983.
[13] P.Dietz and S.R.Kosaraju, Recognition of topological equivalence of patterns by array automata, JCSS 2 (1980), 111-116.
[14] C.R.Dyer, One-way bounded cellular automata, Information and Control 44 (1980),261-281.
[15] __ , Relation of one-way parallel/sequential automata to 2-d finite automata, Information Sciences 23 (1981), 25-30.
[16] C.R.Dyer and A.Rosenfeld, Cellular pyramids for image analysis, University of Maryland, Computer Science Center, TR-544, AFOSR-77-3271, 1977.
[17] __, Parallel image processing by memory-augmented cellular automata, IEEE Trans. on Pattern Analysis and Machine Intelligence, PAMI-3, 1 (1981), 29-41.
[18] __, Triangle cellular automata, Information and Control 48 (1981), 54-69.
[19] E.M.Ehlers and S.H.Von Solms, A hierarchy of random context grammars and automata, Information Sciences 42 (1987), 1-29.
[20] M.J.Fisher, Two characterizations of the context sensitive languages, IEEE Symp. on Switching and Automata Theory (1969), 149-156.
[21] J.Hartmanis, The structural complexity column, Bulletin of the EATCS, 33 (October 1987), 26-39.
[22] A.Hemmerling, Normed two-plane traps for finite systems of cooperating compass automata, EIK 23, 8/9 (1987),, 453-470.
[23] F.Hoffmann, One pebble does not suffice to search plane labyrinths, Lecture Notes in Computer Science 117 (Fundamentals of Computation Theory), 1981, 433-444.
[24] J.E.Hopcroft and J.D.Ullman, Some results on tape-bounded Turing machines, J.Assoc.Comput.Math. 19, 2 (1972), 283-295.
[25] __, Introduction to Automata Theory, Languages, and Computation, Addison-Wesley, Reading, Mass. 1979.
[26] J.Hromkovic, K.Inoue and I.Takanami, Lower bounds for language recognition on two-dimensional alternating multihead machines, To appear in JCSS.
[27] O.H.Ibarra and R.T.Melson, Some results concerning automata on two-dimensional tapes, Intern.J.Computer Math. 4-A (1974), 269-279.
[28] N.Immerman, Nondeterministic space is closed under complement, submitted for publication.
[29] K.Inoue, Investigations of two-dimensional on-line tessellation acceptors (in Japanese), Ph.D.Thesis, Nagoya University, 1977.
[30] K.Inoue and A.Nakamura, Some notes on parallel sequential array acceptors (in Japanese), IECE of Japan Trans.(D), March 1975, 167-169.
[31] __ , On the relation between two-dimensional on-line tessellation acceptors and one-dimensional bounded cellular acceptors (in Japanese), IECE of Japan Trans.(D), September 1976, 613-620.
[32] __, Some properties of one-way parallel sequential array acceptors and two-dimensional one-marker automata (in Japanese), IECE of Japan Trans.(D), September 1976, 682-683.
[33] __, Some properties of parallel sequential array acceptors and two-dimensional two-marker automata (in Japanese), IECE of Japan Trans.(D), September 1976, 680-681.
[34] __, Some properties of two-dimensional on-line tessellation acceptors (in Japanese), IECE of Japan Trans.(D), October 1976, 695-702.
[35] __, Some properties of two-dimensional on-line tessellation acceptors, Information Sciences 13 (1977), 95-121.
[36] __, Some properties of two-dimensional automata with a one-letter alphabet -Recognizability of functions by two-dimensional automata- (in Japanese), IECE of Japan Trans.(D), September 1977, 679-686.
[37] __, Nonclosure properties of two-dimensional on-line tessellation acceptors and one-way

parallel sequential array acceptors, IECE of Japan Trans.(E), September 1977, 475-476.
[38] __, Some properties on two-dimensional nondeterministic finite automata and parallel sequential array acceptors (in Japanese), IECE of Japan Trans.(D), November 1977, 990-997.
[39] __, Two-dimensional multipass on-line tessellation acceptors, Information and Control 41, 3 (June 1979), 305-323.
[40] __, Two-dimensional finite automata and unacceptable functions, Intern.J.Computer Math. Section A, 7 (1979), 207-213.
[41] K.Inoue and I.Takanami, A note on closure properties of the classes of sets accepted by tape-bounded two-dimensional Turing machines, Information Sciences 15, 1 (1978), 143-158.
[42] __, Cyclic closure properties of automata on a two-dimensional tape, Information Sciences 15, 1 (1978), 229-242.
[43] __, A note on bottom-up pyramid acceptors, Information Processing Letters 8, 1 (1979), 34-37.
[44] __, Three-way tape-bounded two-dimensional Turing machines, Information Sciences 17, 3 (1979), 195-220.
[45] __, Closure properties of three-way and four-way tape-bounded two-dimensional Turing machines, Information Sciences 18, 3 (1979), 247-265.
[46] __, Three-way two-dimensional multicounter automata, Information Sciences 19, 1 (1979), 1-20.
[47] __, Some properties of three-way two-dimensional multicounter automata over square tapes (in Japanese), IECE of Japan Trans.(D), October 1979, 673-680.
[48] __, A note on deterministic three-way tape-bounded two-dimensional Turing machines, Information Sciences 20, 1 (1980), 41-55.
[49] __, A note on decision problems for three-way two-dimensional finite automata, Information Processing Letters 10, 5 (July 1980), 245-248.
[50] K.Inoue, I.Takanami and J.Hromkovic, A leaf-size hierarchy of two-dimensional alternating Turing machines, Discrete Algorithms and Complexity, Academic Press (1987), 389-404.
[51] K.Inoue, I.Takanami and A.Nakamura, A note on two-dimensional finite automata, Information Processing Letters 7, 1 (1978), 49-52.
[52] __, Nonclosure property of nondeterministic two-dimensional finite automata under cyclic closure, Information Sciences 22, 1 (1980), 45-50.
[53] __, Connected pictures are not recognizable by deterministic two-dimensional on-line tessellation acceptors, Computer Vision, Graphics, and Image Processing 26 (1984), 126-129.
[54] __, A note on time-bounded bottom-up pyramid cellular acceptors, To appear in Information Sciences.
[55] K.Inoue, I.Takanami and H.Taniguchi, Three-way two-dimensional simple multihead finite automata -Hierarchical properties- (in Japanese), IECE of Japan Trans.(D), February 1979, 65-72.
[56] __, Three-way two-dimensional simple multihead finite automata -Closure properties- (in Japanese), IECE of Japan Trans.(D), April 1979, 273-280.
[57] __, The accepting powers of two-dimensional automata over square tapes (in Japanese), IECE of Japan Trans.(D), February 1980, 113-120.
[58] __, Two-dimensional alternating Turing machines, Theoretical Computer Science 27 (1983), 61-83.
[59] A.Ito, k.Inoue, I.Takanami and H.Taniguchi, Two-dimensional alternating Turing machines with only universal states, Information and Control 55, 1-3 (1982), 193-221.
[60] __, A note on space complexity of nondeterministic two-dimensional Turing machines, IECE of Japan Trans.(E), August 1983, 508-509.
[61] __, Hierarchy of the accepting power of cellular space based on the number of state changes (in Japanese), IECE of Japan Trans.(D), September 1985, 1553-1561.
[62] __, Relationships of the accepting powers between cellular space with bounded number of state-changes and other automata (in Japanese), IECE of Japan Trans.(D), September 1985, 1562-1570.
[63] __, State-change bounded rectangular array cellular space acceptors with three-neighbor (in Japanese), IEICE of Japan Trans.(D), December 1987, 2339-2347.
[64] A.Ito, K.Inoue and I.Takanami, A note on three-way two-dimensional alternating Turing machines, To appear in Information Sciences.
[65] __, Deterministic on-line tessellation acceptors are equivalent to two-way two-dimensional alternating finite automata through 180° rotations, submitted for publication.
[66] __, A relationship between one-dimensional bounded cellular acceptors and two-dimensional alternating finite automata, manuscript (1987).
[67] __, The simulation of two-dimensional one-marker automata by three-way two-dimensional Turing machines, in this issue.
[68] __, Some closure properties of the class of sets accepted by three-way two-dimensional alternating finite automata, submitted for publication.
[69] K.Iwama, ASPACE(o(loglog n)) is regular, Research Report, KSU/ICS, Institute of Computer Sciences, Kyoto Sangyo University, March 1986.
[70] E.B.Kinber, Three-way automata on rectangular tapes over a one-letter alphabet, Information Sciences 35 (1985), 61-77.
[71] S.R.Kosaraju, On some open problems in the theory of cellular automata, IEEE Trans. on Com-

puters C-23, 6 (1974), 561-565.
[72] __, Fast parallel processing array algorithms for some graph problems, Proc. of the 11th Annual ACM Symp. on Theory of Computing (1979), 231-236.
[73] K.Krithivasan and R.Siromoney, Array automata and operations on array languages, Intern.J.Computer Math. 4-A (1974), 3-30.
[74] R.E.Ladner, R.J.Lipton and L.J.Stockmeyer, Alternating pushdown automata, Proc. of the 19th IEEE Symp. on Foundations of Computer Science (1978), 92-106.
[75] C.Meinel, The importance of plane labyrinths, EIK 18, 7/8 (1982), 419-422.
[76] D.L.Milgram and A.Rosenfeld, Array automata and array grammars, IFIP Congress 71 (1971), Booklet Ta2, 166-173.
[77] D.L.Milgram, A region crossing problem for array-bounded automata, Information and Control 31 (1976), 147-152.
[78] K.Morita, Computational complexity in one- and two-dimensional tape automata, Ph.D. Thesis, Osaka University, 1978.
[79] K.Morita and K.Sugata, Three-way horizontally context-sensitive array grammars (in Japanese), Technical Report No.AL80-66, IECE of Japan (1981).
[80] K.Morita, H.Umeo and K.Sugata, Computational complexity of L(m,n) tape-bounded two-dimensional tape Turing machines (in Japanese), IECE of Japan Trans.(D), November 1977, 982-989.
[81] __, Language recognition abilities of several two-dimensional tape automata and their relation to tape complexities (in Japanese), IECE of Japan Trans.(D), December 1977, 1077-1084.
[82] K.Morita, H.Umeo, H.Ebi and K.Sugata, Lower bounds on tape complexity of two-dimensional tape Turing machines (in Japanese), IECE of Japan Trans.(D), 1978, 381-386.
[83] K.Morita, H.Umeo and K.Sugata, Accepting capability of offside-free two-dimensional marker automata -the simulation of four-way automata by three-way tape-bounded Turing machines-(in Japanese), Technical Report No.AL79-2, IECE of Japan, 1979.
[84] A.Nakamura and K.Aizawa, Acceptors for isometric parallel context-free array languages, Information Processing Letters 13, Nos4-5 (1981), 182-186.
[85] A.Nakamura and C.R.Dyer, Bottom-up cellular pyramids for image analysis, Proc. of the 4th Int.Joint Conf.Pattern Recognition, 1978.
[86] K.Nakazono, K.Morita and K.Sugata, Accepting ability of linear time nondeterministic bottom-up pyramid cellular automata (in japanese), IEICE of Trans.(D), February 1988, 458-461.
[87] J.Pecht, T-recognition of T-languages, a new approach to describe and program the parallel pattern recognition capabilities of d-dimensional tessellation structures, Pattern Recognition 19, 4 (1986), 325-338.
[88] A.Rosenfeld, Some notes on finite-state picture languages, Information and Control 31 (1976), 177-184.
[89] __, Picture Languages (Formal Models for Picture Recognition), Academic Press, New York, 1977.
[90] A.Rosenfeld and D.L.Milgram, Parallel/sequential array automata, Information Processing Letters 2 (1973), 43-46.
[91] W.J.Savitch, Relationships between nondeterministic and deterministic tape complexities, JCSS 4 (1970), 177-192.
[92] S.Seki, Real-time recognition of two-dimensional tapes by cellular automata, Information Sciences 19 (1979), 179-198.
[93] S.M.Selkow, One-pass complexity of digital picture properties, J.Assoc.Comput.Math. 19(2) (1972), 283-295.
[94] A.N.Shah, Pebble automata on arrays, Computer Graphics and Image Processing 3 (1974), 236-246.
[95] __, Pushdown automata on arrays, Information Sciences 25 (1981), 175-193.
[96] M.Sipser, Halting space-bounded computatiopns (Note), Theoretical Computer Science 10 (1980), 335-338.
[97] R.Siromoney, Array languages and Lindenmayer systems -a survey, in 'The Book of L' (eds. G.Rozenberg and A.Salomaa), Springer-Verlag, Berlin, 1985.
[98] __, Advances in array languages, Lecture Notes in Computer Science 291 (Graph-Grammars and Their Application to Computer Science), Ehrig et al. (Eds.), 1987, 549-563.
[99] R.Siromoney and G.Siromoney, Extended controlled table L-arrays, Information and Control 35 (1977), 119-138.
[100] A.R.Smith III, Two-dimensional formal languages and pattern recognition by cellular automata, Proc. of the 12th Switching and Automata Theory (1971), 144-152.
[101] __, Real-time language recognition by one-dimensional cellular automata, JCSS 6 (1972), 233-253.
[102] R.E.Stearns, J.Hartmanis and P.M.Lewis II,, Hierarchies of memory limited computations, IEEE Conf.Rec.on Switching Circuit Theory and Logical Design (1965),, 179-190.
[103] R.Szelepcsenyi, The method of forcing for nondeterministic automata, submitted for publication.
[104] A.Szepietowski, A finite 5-pebble automaton can search every maze, Information Processing Letters 15, 5 (December 1982), 199-204.
[105] __, There are no fully space constructible functions between loglog n and log n, Information Processing Letters 24, 6 (April 1987), 361-362.
[106] __, On three-way two-dimensional Turing machines, manuscript (1987).

[107] H.Taniguchi, K.Inoue, I.Takanami and S.Seki, (k,l)-neighborhood template $\mathcal{A}$-type bounded cellular acceptors (in Japanese), IECE of Japan Trans.(D), March 1981, 244-251.
[108] __, (k,l)-neighborhood template $\mathcal{A}$-type bounded cellular acceptors -refinements of hierarchical properties-(in Japanese), IECE of Japan Trans.(D), September 1983, 1062-1069.
[109] __, Relationship between the accepting powers of (k,l)-neighborhood template $\mathcal{A}$-type 1-dimensional bounded cellular acceptors and other types of 2-dimensional automata (in Japanese), IECE of Japan Trans.(D), October 1985, 1711-1718.
[110] __, Closure properties of (k,l)-neighborhood template $\mathcal{A}$-type 1-dimensional bounded cellular acceptors (in Japanese), IECE of Japan Trans.(D), March 1986, 279-290.
[111] K.Taniguchi and T.Kasami, Some decision problems for two-dimensional nonwriting automata (in Japanese), IECE of Japan Trans.(C), 1971, 578-585.
[112] M.Toda, K.Inoue and I.Takanami, Two-dimensional pattern matching by two-dimensional on-line tessellation acceptors, Theoretical Computer Science 24 (1983), 179-194.
[113] H.Umeo, K.Morita and K.Sugata, Pattern recognition by automata on a two-dimensional tape, IECE of Japan Trans.(D), November 1976, 817-824.
[114] R.Vollmar, On cellular automata with a finite number of state changes, Comput.Suppl. 3 (1981),181-191.
[115] P.S.P.Wang, Finite-turn repetitive checking automata and sequential/parallel matrix languages, IEEE Trans. on Computers C-30, 5 (May 1981), 366-370.
[116] Y.Yamamoto, K.Morita and K.Sugata, Space complexity for recognizing connectedness in three-dimensional patterns, IECE of Japan Trans.(E), 1981, 778-785.

IMYCS'88, Smolenice Castle, November 14-18, 1988.

Complexity Theory and Formal Languages

Klaus-Jörn Lange
Institut für Informatik
Technische Universität München
Arcisstr. 21
D-8000 München 2

I. Introduction

This lecture contains a collection of restults relating and connecting formal languages with complexity theory. It is one aim of this work to show, how complexty theory serves as a unifying framework integrating many approaches and results which seem to be unrelated at first sight. Thus we are interested in exploring and crossing the border between the theory of formal languages and complexity theory as presented in [48]. (We do not deal here with algorithms, communication complexity, VLSI systems, or relativizations.)

We assume the reader to be familiar with the basic notions of formal languages and complexity theory as contained in [23]. A very detailed survey of complexity theory is given in the excellent book [66] of Wagner and Wechsung in 1986. In particular, we will use without explanation the following formalisms:

- off-line, multitape turing machines and their complexity measures
- determinism, nondeterminism, and alternation
- the Chomsky hierarchy, OL systems, macro-languages, and stack automata
- the o( ) and O( ) notation.

Just to avoid confusions, we specify the notation /v/ for the length of a word v and λ for the empty word.

This paper is divided into four parts. The first section is this introduction. Part II contains some aspects of sequential complexity theory. Section III, the main part of this work, describes some connections between formal languages and complexity theory. Finally Part IV collects some recent results of parallel complexity theory.

II. Sequential complexity

The beginning of complexity theory can be determined by the discovery of arbitrarily hard sets; that is for every computable function f there exists a set L such that every algorithm A solving the word problem of L takes at least time f(/v/) for infinitely many v. This approach, to say $L \in Dtime(f(n))$ if L is recognizable within f(/v/) steps for almost all inputs v by some algorithm for L, is called the worst case approach and is the most frequently used method of complexity theory compared with average case analysis or generally stochastic methods.

In this way there was a phase of complexity theory, in which one tried to determine the 'absolute complexity' of a problem, i.e. given a problem P, find a time bound f such that $P \in Dtime(f)$ and $P \notin Dtime(g)$ for any $g \in o(f)$. A statement of the type $P \in Dtime(f)$ gives an upper bound for the running time necessary to solve p and is usually proved by exhibiting an algorithm for P running in time f. The problem is to prove lower bounds, i.e. statements of the type $P \notin Dtime(g)$. Until now there is no general theory or systematic method to obtain lower bounds. There exist some single results which pertain to special sets, under more or less restricting assumptions with a more or less restricted machine model.
A way to cope with this situation is to compare the complexity of problems by using reducibilities and completeness.

We call a mapping $f: X^* \to Y^*$, where X and Y are finite alphabets, 'LOGSPACE computable' (resp. 'POLTIME computable'), if there is a Turing-Machine M wich, started with some $v \in X^*$ on its read only input tape, computes and prints f(v) on its write only output tape, consuming no more than O(log(/v/)) space on its working tapes (resp. performing no more than p(/v/) steps for some polynomial p).

A set $L \subseteq X^*$ is many-one 'LOG-reducible' to a set $M \subseteq Y^*$, denoted by $L \leq_m^L M$, if there exists a LOGSPACE computable mapping $f: X^* \to Y^*$ such that $v \in L$ if and only if $f(v) \in M$ holds for all

$v \in X^*$. If we require f to be POLTIME computable, instead, we get 'POL-reducibility' denoted

by $L \leq_m^P M$. If not otherwise stated we will use in the following LOG-reducibilities and drop the prefix 'LOG'. The 'LOG-closure' of a class of languages $\boldsymbol{A}$ is denoted by

$LOG(A) := \{L \mid \exists M \in A : L \leq_m^L M\}$.

Proposition 2.1

For every class $\boldsymbol{A}$ we have $LOG(\text{Co-}\boldsymbol{A}) := \text{Co-}LOG(\boldsymbol{A})$ and $LOG(LOG(\boldsymbol{A})) = LOG(\boldsymbol{A})$.

Here $\text{Co-}\boldsymbol{A} := \{X^* \setminus L \mid \exists L, X : L \subseteq X^*, L \in \boldsymbol{A}\}$.

Thus we try to determine the 'relative complexity' of a problem, i.e. given a problem P find a class $\boldsymbol{A}$ such that $P \in \boldsymbol{A}$ and P is A-complete. Here $P \in \boldsymbol{A}$ is a 'relative upper bound' and the $\boldsymbol{A}$–completeness of P is a 'relative lower bound' for the complexity of P. A Motivation for analysing relative complexities is that a relative lower bound becomes an absolute one, as soon as the corresponding comparision of complexity classes is solved: if a problem P is complete for a class A and if $\boldsymbol{A}$ is not contained in DTIME(f(n)), then for some $c>0$ $P \notin DTIME(f(n^c))$. Unfortunately, the status of very many important class inclusions is unknown. Even worse, 'relativization result' (essentially this means considering time or tape-bounded Turing reducibilities instead of complexity classes) indicate that nearly all known proof methods cannot answer these questions ([7]). Nevertheless, completeness results are a valuable tool to characterize complexities of problems by infering class properties to complete problems.

Typical classes in complexity theory are $\boldsymbol{PSPACE} := \cup k \geq 1 : DSPACE(n^k)$, $\boldsymbol{NP} := \cup k \geq 1 : NTIME(n^k)$, $\boldsymbol{P} := \cup k \geq 1 : DTIME(n^k)$, $\boldsymbol{NL} := NSPACE(\log n)$, and $\boldsymbol{L} := DSPACE(\log n)$. These classes form a hierarchy: $\mathbf{L \subseteq NL \subseteq P \subseteq NP \subseteq PSPACE}$. Although most researchers believe this hierarchy to be proper, i.e. that all inclusions are strict, $\boldsymbol{NL \neq PSPACE}$ is the only known inequality here. Hence, equalities like $\boldsymbol{L=NP}$ or $\boldsymbol{P=PSPACE}$ still may hold. All these classes are closed under LOG-reducibility, i.e. $LOG(\mathbf{PSPACE}) = \mathbf{PSPACE}, \ldots, LOG(\mathbf{L}) = \mathbf{L}$, and possess well known complete sets:

- **True quantified boolean formulae:** the set *TQBF* of all true quantified boolean formulae is ***PSPACE***-complete ([57]),
- **Satisfiability:** the set *SAT* of all satisfiable (=true existentially quantified) boolean formulae is ***NP***-complete ([13]),
- **Circuit value problem:** the set *CVP* of all pairs of a boolean circuit and an input assignement such that a designated output yields the value true is ***P***-complete ([34]),

- **Graph-accessability problem:** the set *GAP* of all directed graphs with designated source and sink such that there exists a path from source to sink is ***NL***-complete ([54]),

and

- the trivial set {1} is ***L***-complete.

An important research topic of complexity theory is the polynomial hierarchy as defined in [46] by oracle machines. It can be characterized by bounded quantification and by alternating Turing machines (see [10], [57], [68]): A set L is an element or the k.-th level of the polynomial hierarchy, denoted by Σ_n^P, if there exist a set $M \in P$ and a polynomial p such that $L=\{v \mid \exists x_1, /x_1/\leq p(/v/): \forall x_2, /x_2/\leq p(/v/): \ldots Qx_k, /x_k/\leq p(/v/): (v,x_1,\ldots,x_k)\in M\}$, where Q denotes $\exists$ for odd k, and $\forall$ otherwise. Further on, set $\Pi_k^P := \text{Co-}\Sigma_k^P$. The classes Σ_k^P and Π_k^P are closed under LOG-reducibility and possess complete sets, which are related to *TQBF* (see [68]). It is an open question whether the polynomial hierarchy is proper, i.e. infinite, which is assumed by most researchers.

As an attempt to construct a LOGSPACE bounded analogy, Chandra et al. defined in [10] the logarithmic alternation hierarchy with the help of alternating Turing machines. To put a space bound on oracle machines or bounded quantification is not abvious and led at first back to the polynomial hierarchy (see [36], [57], [40]). Later on, a logarithmic quantification hierarchy and a logarithmic oracle hierarchy were found. But while the former coincides with the logarithmic alternation hierarchy ([41]), the later contains the whole logarithmic alternation hierarchy in its second level ([52]). Again, the common believe assumed these hierarchies to be proper, at least until 1986. But since then things changed dramatically: in October 1986 the logarithmic alternation hierarchy collapsed to its second level ([43]). Then in April 1987 this result was improved by Schöning and Wagner in [55]. They showed that the second levels of the logarithmic alternation hierarchy and the logarithmic oracle hierarchy coincide, thereby collapsing the logarithmic oracle hierarchy to its second level. Finally, in July 1987 Immerman showed the closure under complement of nondeterministic space classes ([27])! Hence the hierarchies mentioned above all collapse to their first level ***NL*** = Co-***NL***. Surprisingly, this result was found independently in April 1987 by R. Szelepcsényi ([62]) from Bratislava, but get not public before late 1987.

III. Formal languages

This section contains a collection of results from the interface between formal language theory and complexity. (More details and related results concerning this topic may be found in [48] or

[66]). Out of the huge abundance of formal languages we selected the Chomsky-languages, the context-free families of Lindenmayer languages, the index and macro languages, and some classes of stack languages. The complexities of several dedidable proplems for these families have been characterized by showing these problems to be complete for well-known complexity classes. In this way complexity theory serves as a unifying framework and relates different families of formal languages thereby giving new insights and a better understanding of formal language theory.

This section is divided into three parts. First of all, we consider the membership problem for classes of languages. We do not consider the general membership problem, i.e. the case that the language generating grammar is not fixed, but part of the input. Very often it is equivalent to the emptiness problem, which is investigated in the second part of this section. Finally, we consider shortly the complexities of operations on formal languages.

1) The membership problem

In the following the families of the context-sensitive, deterministic context-sensitive, context-free, deterministic context-free, linear, and regular languages are denoted by ***CS***,***DLBA***, ***CF***, ***DCF***, ***LIN*** and ***REG***, respectively. The type 0 languages, i.e. the recursive enumerable languages, are not considered, since they strictly contain the decidable languages, which in turn are a proper superset for any complexity class.

Since linear bounded automata correspond to linear space bounded turing machines we have ***CS*** = NSPACE(n) and ***DLBA*** = *DSPACE(n)*. Applying reducibilities of the type f(v) = v#p(/v/), where p is a polynomial, we get

Proposition 3.1

LOG(***CS***) = LOG (***DLBA***) = ***PSPACE***.

Concerning context-free languages we know by [69] ***CF*** $\subseteq$ DTIME (n^3).

Proposition 3.2

LOG(***CF***) $\subseteq$ ***P***.

Now let DAPDA(log n) (resp. NAPDA(log n)) be the class of languages accepted by deterministic (resp. nondeterministic) push-down automata with a two-way input tape augmented with a logarithmically space bounded working tape. The following result of Cook implies proposition 3.2.

Theorem 3.3 ([12])

DAPDA(log n) = NAPDA(log n) = ***P***.

Sudborough characterized the difference between LOG(**CF**) and ***P*** by putting a polynomial bound on the running time of augmented push-down automata, defining the classes $DAPDA_{PT}(\log n)$ and $NAPDA_{PT}(\log n)$:

Theorem 3.4 ([61])

$LOG(\boldsymbol{DCF}) = DAPDA_{PT}(\log n)$ and $LOG(\boldsymbol{CF}) = NAPDA_{PT}(\log n)$.

The best space bound for recognition of context-free languages is still given by the algorithm of Lewis, Stearns, and Hartmanis:

Theorem 3.5 ([45])

$LOG(\boldsymbol{CF}) \subseteq DSPACE(\log^2 n)$.

Since ***P*** and $DSPACE(\log^2 n)$ seem to be incomparable, ***CF*** should not be expected to be complete either for ***P*** or $DSPACE(\log^2 n)$. Concerning simultaneous resource bounds, Cook showed

Theorem 3.6 ([14])

$LOG(\boldsymbol{DCF}) \subseteq DTIMESPACE(POL, \log^2 n)$,

where $DTIMESPACE.(f,g)$ (resp. $NTIMESPACE(f,g)$) denotes the class of all languages recognizable be deterministic (resp. nondeterministic) Turingmachines with g-bounded workingtapes and a runningtime bounded by f and POL means the union over all polynomials.

For context-free languages only $LOG(\boldsymbol{CF}) \subseteq NTIMESPACE(POL, \log^2 n)$ is known.

The families of the deterministic context-free languages and of the linear languages are incomparable and the same seems to hold for their LOG-closures, since Sudborough was able to show

Theorem 3.7 ([59])

$LOG(\boldsymbol{LIN}) = \boldsymbol{NL}$

Some remarks:

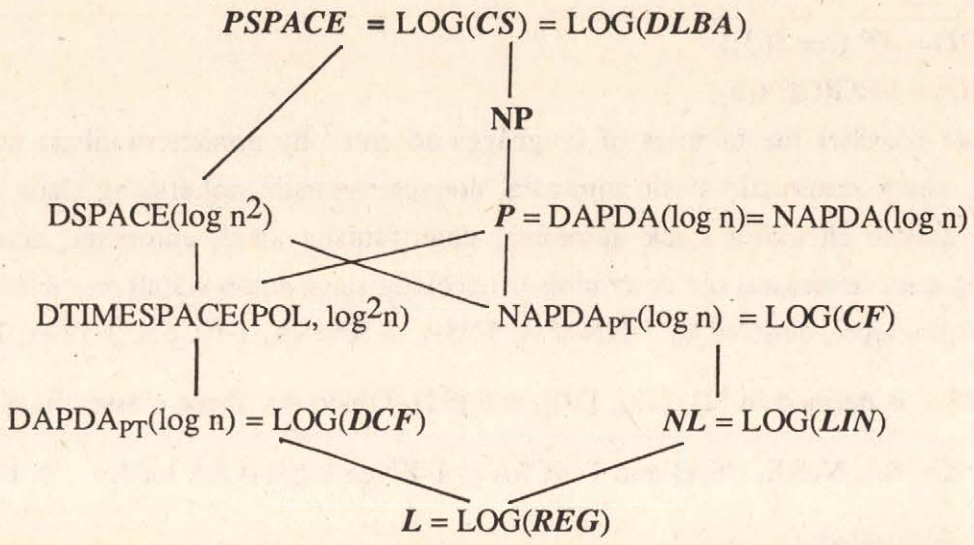
i) The same result holds true for the family of nondeterministic one-counter languages.

ii) Theorems 3.7 and 3.5 imply the famous result of Savitch $NSPACE(f) \subseteq DSPACE(f^2)$ for $\log(n) \in O(f(n))$ ([54]). It is remarkable that the algorithm of Lewis, Stearns, and Hartmanis is considerably older than that of Savitch.

iii) The result of Immerman-Szelepcsényi implies with the help of proposition 2.1: $LOG(\text{Co-}\boldsymbol{LIN}) = LOG(\boldsymbol{LIN})$.

Finally, we have $LOG(\boldsymbol{REG}) = \boldsymbol{L}$ because of $\boldsymbol{REG} \subseteq \boldsymbol{L}$.

The results of this part are summarized in the following diagram:



We see that Chomsky languages fit in nicely in the existing framework of complexity classes. The same holds true for the many types of Lindenmayer Languages. Among their wide variety we cite result characterizing the complexities of languages generated by the following classes of context-free L-systems. ***ETOL, EOL, EDTOL,*** and ***EDOL***. (The definitions of these families of languages may be found e.g. in ([51]).

Theorem 3.8

a) LOG(***ETOL***) = ***NP*** ([63]),

b) LOG(***EOL***) = LOG(***CF***) ([60]),

c) LOG(***EDTOL***) = ***NL*** ([32]), and

d) LOG(***EDOL***) = ***L*** ([60]).

Remark: i) although **EOL** and **EDTOL** are incomparable, Theorem 1.8 indicates that EOL languages might have more difficult parsing problems than EDTOL languages. This relation turns, if we consider the trio's, i.e. the closure under nonerasing homomorphisms, inverse homomorphisms, and intersections with regular sets, *M*(***EOL***) and *M* (***EDTOL***) generated by ***EOL*** resp. ***EDTOL***. We then have LOG (*M*(***EOL***)) = LOG(***CF***) and LOG(*M*(***EDTOL***)) = ***NP*** (see [39]). Note that both **EOL** and **EDTOL** are closed under homomorphisms and intersections with regular sets.

ii) There are a lot of results examining the complexities of controlled L systems and more general parallel rewriting mechanisms (see e.g. [4], [5] or [38]).

The family ***Index*** of indexed languages by Aho ([1]) was shown by Fischer to coincide with the family ***OI*** of outside in macro languages ([17]). Fischer also defined the inside-out macro languages, constituting the family ***IO***, and showed these two classes to be incomparable. Complexity results indicate that ***OI*** seems to have harder membership problems than ***IO.***

Theorem 3.9

a) LOG(***OI***) = ***NP*** (see [63])

b) LOG(***IO***) = LOG(***CF***)([6]),

Finally, we consider the families of languages accepted by nondeterministic nestedstack automata, nondeterministic stack automata, nondeterministic nonerasing stack automata, nondeterministic checking stack automata, deterministic stack automata, deterministic nonerasing stack automata, and deterministic checking stack automata(all restricted to have a one-way input tape), denoted by 1-NNstSA, 1NSA, 1-NNeSA, 1-NCSA, 1-DSA, 1-DNeSA, and 1-DCSA as defined in [2], [19], [20], and [21]. Obviously these classes fulfill 1-DX $\subseteq$ 1-NX for X$\in$ {SA, NeSA, CSA} and 1-XCSA $\subseteq$ 1-XNeSA $\subseteq$ 1-XSA for X$\in$ {D, N}, as well as 1-NSA $\subseteq$ 1-NNstSA.

By 1-NNstSA = ***INDEX*** ([2]) and the existence of ***NP***-complete stes in 1-NCSA ([56]) we get

Theorem 3.10

LOG(1-NCSA)=LOG(1-NNeSA)=LOG(1-NSA)=LOG(1-NNstSA)=***NP***

and for the deterministic case

Theorem 3.11

a) LOG(1-DSA) $\subseteq$ ***P*** ([19])

b) LOG(1-DCSA)=***L***([25]).

Remarks: i) for the automata types occuring in this subsection also alternating versions exist (see e.g. [10], [29], [35], [37]).

ii) Since all the families of formal languages occuring in this subsection are contained in **NP**, a natural question to ask is: are there any 'reasonable' families of formal languages with a decidable emptiness problem and membership problems which are likely to lie outside of ***NP***. (An 'unreasonable' family of this type would be ***HFIL***, the class of all homomorphic images of languages generated by ***IL*** systems with a finite, possibly empty set of axioms: On the one hand the emptiness problem of ***HFIL*** is simple, since we only have to check the set of axioms for emptiness, while on the other hand ***HFIL*** coincides with the set or all recursively enumerable languages!).

2) The emptiness problem

For a class ***X*** of languages given by some language generating devices of type ***X*** we denote by Ø-***X*** the nonemptiness problem of this class: given an object A of type ***X*** determine whether A generate does not the empty set. There might be difficulties if ***X*** is represented by several mechanisms with different properties (see e.g. remark ii) at the end of the first subsection). But in our case this only affects the class ***X*** = ***REG***, where we assume that recular sets are given by

(non)deterministic finite automata and not by regular expressions, i.e. the types of language generating devices are DFA and NFA.

In these emptiness problems all occuring automata are assumed to have an input tape with one one-way reading head. (Two-way or two-head automata in nearly all cases have an undecidable emptiness problem. Thus the emptiness problem of CS is undecidable). There is a very interesting connection between the complexities of the emptiness problem for one-way automata and the membership problem of two-way automata (as already noted by Hunt in [24]) in that the complexity of the emptiness problem for one-way automata of type X very often coincides with the cmplexity of languages accepted by two-way automata of type X. This transition from the one-way to the two-way case forms an important bridge between the theory of formal languages and complexity theory.

Since we are interested in LOG-closures of language classes it is reasonable to investigate two-way automata equipped with additional memory of logarithmic size (or equivalently two-way multi-head automata). Thus we consider classes $\boldsymbol{NAX}(\log n)$ and $\boldsymbol{DAX}(\log n)$ denoting the families of languages accepted by nondeterministic resp. deterministic two-way automata of type X augmented with a logarithmically bounded working tape (see [12]).

These classes are closed under LOG-reducibilities, i.e. $\mathrm{LOG}(\boldsymbol{NAX}(\log n)) = \boldsymbol{NAX}(\log n))$ and $\mathrm{LOG}(\boldsymbol{DAX}(\log n)) = \boldsymbol{DAX}(\log n)$.

The context-free languages are a good example for the mentioned connection between emptiness problems and membership in two-way languages.

Theorem 3.12

a) $(\ \mathrm{LOG}\ (Ø\text{-}\boldsymbol{CF}) = \mathrm{LOG}(Ø\text{-}\boldsymbol{DCF}) = \boldsymbol{P}$ ([31]),

b) $\boldsymbol{NAPDA}(\log n) = \boldsymbol{DAPDA}\ (\log n) = \boldsymbol{P}$ ([12]).

Theorem 3.13

a) $\mathrm{LOG}(Ø\text{-}\boldsymbol{LIN}) = \mathrm{LOG}(Ø\text{-}\boldsymbol{NFA}) = \mathrm{LOG}(Ø\text{-}\boldsymbol{DFA}) = \boldsymbol{NL}$ ([30]),

b) $\boldsymbol{NAPDA}_{1\text{-turn}}(\log n) = \boldsymbol{NL}$ ([26]),

c) $\boldsymbol{NAFA}(\log n) = \boldsymbol{NL}$ and $\boldsymbol{DAFA}(\log n) = \boldsymbol{L}$ (obvious).

Here $PDA_{1\text{-turn}}$ denotes push-down automata, which make at most one reversal on their push-down.This (one-way) device exactly recognizes the linear languages.

Concerning OL systems we remark that $\boldsymbol{ETOL}$ (resp. $\boldsymbol{EOL}$) coincides with the class of languages recignized by CSPD (resp. RPAC) automata ([64] resp. [50]). We then have

Theorem 3.14

a) $\mathrm{LOG}\ (Ø\text{-}\boldsymbol{ETOL}) = \mathrm{LOG}(Ø\text{-}\boldsymbol{EDTOL}) = \boldsymbol{NACSPD}(\log n) = \boldsymbol{PSPACE}$ ([33], 64])

b) $\boldsymbol{NP} \subseteq \mathrm{LOG}(Ø\text{-}\boldsymbol{EOL}) = \boldsymbol{NARPAC}(\log n) \subseteq \boldsymbol{PSPACE}$([44]),

c) $\mathrm{LOG}(Ø\text{-}\boldsymbol{EDOL}) = \mathbf{NP}$([33]).

Open question: is it possible to locate ***Ø-EOL*** in the polynomial hierarchy?

Theorem 3.15 ([24])

LOG(***Ø-OI***) = ***EXPOLYTIME***: = $\cup k>o$: DTIME(2^{nk}).

Remark: The algorithm to solve ***Ø-IO*** in [17] and [1] runs in polynomial time (as mentioned in [24]) but is not correct. Else theorem 3.14 would imply ***P*** = ***PSPACE***. A correct algorithm may be found in [3]. ***Ø-IO*** should be expolytime-complete, too.

All types of stack automata but the checking stack show the close connection between emptiness and two-way membership mentioned above.

Theorem 3.16

a) LOG(***Ø-NNstSA***) = LOG(***Ø-NSA***) = LOG(***Ø-DSA***) = ***EXPOLYTIME*** ([24]),

b) ***NANstSA***(LOG N) = ***NASA***(log n) = ***DASA***(log n) = ***EXPOLYTIME*** ([8], [25]),

c) LOG(***Ø-NNeSA***) = LOG(***Ø-DNeSA***) = ***PSPACE*** ([24]),

d) ***NANeSA***(log n) = ***DANeSA***(log n) = ***PSPACE*** ([25]),

e) ***NACSA***(log n) = ***PSPACE*** ([25]),

f) ***DACSA***(log n) = ***L*** ([25]).

Remarks: i) ***Ø-DCSA*** is ***NL***-hard by Theorem 2.2.

ii)The checking stack automata is one of the few models in complexity theory, where determinism and nondeterminism are provably in equivalent.

3) Operations on formal languages

This subsection gives a short survey on results, characterizing complexity classes by formal language operators. We consider the following operations on families of formal languages:

i) Kleene closure: A^*: = $\{L^* | L \in A\}$,

ii) nonerasing homomorphisms: $H_\lambda(A)$: = {h(L)|L∈***A***, h is a nonerasing homomorphism},

iii) iterated shuffle (see e.g. [28]): $A^\dagger$: = $\{L^\dagger | L \in A\}$.

Further on, let 1-***L*** be the set of all languages accepted by LOGSPACE turing machines with an one-way input tape.

Theorem 3.17 ([18], [40], [47], [67])

i) ***NL*** = LOG(***L****),

ii) ***NL*** = LOG(H_λ(1-***L***)),

iii) ***NP*** = LOG(H_λ(***L***)),

iv) ***NP*** = LOG($L^\dagger$).

Remarks:

i) ***NL*** and ***NP*** are characterized by these operations since ***NL*** closed under * and ***NP*** in addition under H_λ and †.

ii) For arbitrary homomorphisms H(**L**) coincides with the class all recursively enumerable languages, which in turn is closed under λ.

Theorem 3.17 suggest the question for language operations chracterizing complexity classes like ***P***, ***PSPACE***, ***EXPOLYTIME*** etc. This question seems to be related to the search for families of formal languages, which are complete for these complexity classes. Concerning the complexity class LOG(***CF***), a modification of iterated insertion yields an operation ^{B} fulfilling $LOG(\boldsymbol{CF}) = LOG(\boldsymbol{L}^B)$ and $LOG(\boldsymbol{CF})^B \subseteq LOG(\boldsymbol{CF})$ (see [42]).

Theorem 3.17 is interesting in particular for giving the possibility to characterize several ways of relativizing complexity classes. We state this result without explaining the details, but refer to [40] instead.

Theorem 3.18

for arbitrary language class ***A*** we have

i) (Ruzzo - Simon - Tompa relativization): $\boldsymbol{NL}\langle\boldsymbol{A}\rangle = LOG(\mathbf{L}(\boldsymbol{A})^*)$

ii) (Ladner-Lynch relativization): $\boldsymbol{NL}(\boldsymbol{A}) = LOG(H_\lambda(1\text{-}\boldsymbol{L}(\boldsymbol{A})))$,

iii) $\boldsymbol{NP}(\boldsymbol{A}) = LOG(H_\lambda(\boldsymbol{L}(\boldsymbol{A})))$.

IV. Parallel Complexity

A very fast growing area of complexity theory is the field of synchronous prallel computations. ([15]) gives a very good summary of this topic). This section is to give some of those results which concern formal languages.

In spite of the many different models of complexity theory for describing parallel computations, there occurs as a rule the phenomenon of parallel time bounded complexity classes being polynomially related to sequential space bounded complexity classes. This led to the declaration of the *Parallel Computation Thesis*, that every reasonable model of parallel computation shows this behaviour (see [57] and [22]).

In the following we consider three main models of parallel complexity theory:

- parallel random access machines with several ways of settling read and write conflicts. In particular we are interested in CRCW-,CREW-, and CROW-PRAM's (see [16]).
- Boolean circuits with bounded, semi-unbounded, or unbounded FAN-IN (see [15] [65]).
- Alternating turing machines ([10]) bounded in time, space, or depth of alternation.

The close relation between these apparently different devices can be seen by the three characterizations of the class AC^1 (see [15]) by

i) alternating LOGSPACE turing machines the alternation depth of which is logarithmically bounded,
ii) unbounded FAN-IN circuits of logarithmic depth, and
iii) logarithmically time-bounded CRCW-PRAM's

We state this as

Theorem 4.1 ([11], [15], [58])

AC^1 := ASPACEDEPTH(log n, log n) = UNBOUNDEDSIZEDEPTH(POL, log n) = CRCW-PRAMTIME(log n).

Here POL again denotes the union over all polynomicals. (Actually, instead of log n often it should read $\cup\, c > 0 : c \cdot \log n$) by Venkateswaran ([65]) LOG(***CF***) coincides with the class of all languages recognizable by uniform semiunbounded circuits of polynomial size and logarithmic depth. (Here semi unbounded means that the fan-in of all and-Gates is bounded by some constant):

Theorem 4.2 ([65])

LOG(***CF***) = SEMIUNBOUNDEDSIZEDEPTH(POL, log n).

This reproves the known fact $\boldsymbol{CF} \subseteq AC^1$, i.e. that context-free languages can be recognized in logarithmic time on a CRCW-PRAM.

Following the ideas of Immerman-Szelepcsényi and using Theorem 2 Borodin et.al. succeeded in showing the closure of LOG(***CF***) under complement.

Theorem 4.3 ([9])

Co-LOG(***CF***) = LOG(***CF***).

Remark: It is possible to show this result directly without using Boolan circuits (see [42]).

The family ***UCF*** of unambiguous context-free language can be recognized in logarithmic TIME on a CREW-PRAM.

Theorem 4.4 ([53]):

LOG(*UCF*) $\subseteq$ CREW-PRAMTIME (log n).

Here the following questions seem to be interesting:

- Is LOG(*UCF*) closed under complement?
- Can these classes be characterized in terms of Boolean circuits or sequential automata?

That polynomial time bounded DAPDA(log n)'s are not an answer to the last question, is indicated by the following surprising equality:

Theorem 4.5 ([16])

LOG(*DCF*) = CROW-PRAMTIME (log n).

It should be remarked, that Theorem 3.6 can be phrased in terms of parallel complexity, since DTIMESPACE (POL,$\log^2$n) = BOUNDEDSIZEWIDTH (POL, $\log^2$ n) ([49]).

Question: Are there similar parallel representations of the other formal languages mentioned in subsection II.1?

References:

[1] A.Aho: Indexed grammars - an extension of context-free grammars, J. Assoc. Comp. Mach. **15** (1968), 647-671.

[2] A.Aho: Nested stack automata, J. Assoc. Comput. Mach. **16** (1969), 383-406.

[3] J. Albert: Über indizierte und m-Block-indizierte Grammatiken, Dissertation, Universität Karlsruhe, 1976, (in German).

[4] P. Asveld: Iterated Context-Independent Rewriting, Ph.D. thesis, Technische Hogeschool Twente, 1978.

[5] R. Asveld: Space-bounded complexity classes and iterated deterministic substitutions, Inform. and Control **44** (1980), 282-299.

[6] P. Asveld: Time ans space complexity of inside-out macro languages, Internat. J. Comput. Math. **10** (1981), 3-14.

[7] T. Baker, J. Gill, R. Solovay: Relativizations of the P = ? NP question, SIAM J. Comp. **4** (1975), 431-442.

[8] C. Beeri: Two-way nested stack automata are equivalent to two-way stack automata, J. Comp. System Sci. **10** (1975), 317-339.

[9] A. Borodin, S. Cook, P. Dymond, W. Ruzzo, M. Tompa: Two applications of complementation via inductive counting, Technical Report 87-10-01, Department of Computer Science, University of Washington, 1987

[10] A. Chandra, D. Kozen, L. Stockmeyer: Alternation, J. Assoc. Comput. Mach. **28** (1981), 114-133.

[11] A. Chandra, L. Stockmeyer, U. Vishkin: Complexity theory for unbounded FAN-IN Parallelism, Proc. of 23rd FOCS (1982), 1-13.

[12] S. Cook: Characterizations of pushdown machines in terms of time-bounded computers, J. Assoc. Comput. Mach. **18** (1971), 4-18.

[13] S. Cook: The complexity of theorem proving procedures, Proc. of the 3rd Annual ACM Symp. on Theory of Computing, 1971, 151-158.

[14] S. Cook: Deterministic CFL's are accepted simultaneously in polynomial time and log squared space, Proc. of the 11the Annual ACM Symp. on Theory of Computing, 1979, 338-345.

[15] S. Cook: A taxonomy of problems with fast parallel algorithms, Information and Control **64** (1985), 2-22.

[16] P. Dymond, W. Ruzzo: Parallel RMS with owned global memory and dterministic context-free language recognition, Proc. of 13th ICALP (1987), Springer LNCS 233, 95-104.

[17] M.J. Fischer, Grammars with Macro-like Production, Ph.D. thesis, Harvard Univ., 1968.

[18] P. Flajolet, J. Steyaert: Complexity of classes of languages and operators, IRIA Laboria, Rap. de Recherche No. 92, Nov. 1974.

[19] S. Ginsburg, S. Greibach, M. Harrison: Stack automata and compliling, J. of Assoc. Comp. Mach. **14** (1967) 172-201.

[20] S. Ginsburg, S. Greibach, M. Harrison: One-way stack automata, J. of Assoc. Comp. Mach. **14** (1967), 389-418.

[21] S. Greibach: Checking automata and one-way stack languages, J. Comp. System Sci. **3** (1969), 196-217.

[22] J.-W. Hong: On similarity and duality of computation, Inform. and Control **62** (1984), 109-128.

[23] J. Hopcroft, J. Ullman: Introduction to Automata Theory, Languages, and Computation, Addison-Wesley, Reading Mass., 1979.

[24] H. Hunt: On the complexity of finite, pushdown, and stack automata, Math. Systems Theory **10** (1976), 33 - 52.

[25] O. Ibarra: Characterizations of some Tape and Time Complexity Cla of Turingmachines in Terms of Multihead and Auxiliary Stack Automata, J.Comp. Systems Sci. **5** (1971), 88 - 117.

[26] O. Ibarra, S. Kim, L. Rosier: Space and time efficient simulations and characterizations of some restricted classes of PDAs, Proc. of the 11th EATCS International Colloquium on Automata, Languages and Programming, 1984, Springer LNCS **172**, 247 - 259.

[27] N. Immerman: Nondeterministic space is closed under complement, Technical Report 552, Yale University, 1987.

[28] M. Jantzen: The power od synchronizing operations on strings, Theoret. Comput. Sci. **14** (1981), 127 - 154.

[29] B. J. Jenner, B. Kirsig: Characterizing the polynomial hierarchy by alternating auxiliary pushdown automata, Proc. of 5th STACS (1988), Springer LNCS **294**, 118 - 125.

[30] N. Jones: Space-bounded reducibility among combinatorial problems, J. Comput. System Sci. **11**, (1975), 68 - 85.

[31] N. Jones, W. Laaser: Complete problems for deterministic polynomial time, Theoret. Comput. Sci. **3**, (1977), 105 - 117.

[32] N. Jones, S. Skyum: Recognition of deterministic ETOL languages in logarithmic space, Inform. and Control **35**, (1977), 177 - 181.

[33] N. Jones, S. Skyum: Complexity of some problems concerning L systems, Math. Systems Theory **13**, (1979), 29 - 43.

[34] R. Ladner, The circuit value problem is log space complete for P, SIGACT NEWS **7**, (1975), 18 - 20.

[35] R. Ladner, R. Lipton, L. Stockmeyer: Alternating pushdown and stack automata, SIAM J. Comput. **13** (1984), 135 - 155.

[36] R. Ladner, N. Lynch: Relativization of questions about log space computability, Math. Systems Theory **10** (1976), 19 - 32.

[37] R. Ladner, L. Stockmeyer, R. Lipton: Alternation bounded auxiliary pushdown automata, Inform. and Control **62** (1984), 93 - 108.

[38] K.-J. Lange: Context-free controlled ETOL systems, Proc. of 9th ICALP 1983, Springer LNCS **154** , 723 - 733.

[39] K.-J. Lange: L Systems and NLOG-Reductions, in 'The book of L', G. Rozenberg and A. Salomaa (Eds.), Springer Verlag, Berlin, 1986, 245 - 252.

[40] K.-J. Lange: 'Nichtdeterministische Reduktionen und logarithmische Hierarchien', Technical Report No. 119, 1986, Fachbereich Informatik, Universität Hamburg, (in German).

[41] K.-J. Lange: Two characterizations of the logarithmic alternation hierarchy, Proc. of MFCS 86 (1986), Springer LNCS **233**, 518 - 526.

[42] K.-J. Lange: On the complexity of iterated insertion, In preparation, 1988.

[43] K.-J. Lange, B. Jenner, B. Kirsig: The logarithmic alternation hierarchy collapses': $A\Sigma_2^L = A\Pi_2^L$', Proc. of 14th ICALP (1987), Springer LNCS **267**, 531 - 541.

[44] K.-J. Lange, M. Schudy: A further ling between formal languages and complexity theory, EATCS Bull. **33** (1987), 67 - 71.

[45] P. Lewis, R. Stearns, J. Hartmanis: Memory Bounds for recognition of context-free and context-sensitive languages, Proc. 6th Annual IEEE Symp. on Switching Circuit Theory and Logical Design, 1965, 191 - 209.

[46] A. Meyer, L. Stockmeyer: The equivalence problem for regular expressions with squaring requires exponential space, Proc. of the 13th Annual IEEE Symposium on Switching and Automata Theory 1972, 125 - 129.

[47] B. Monien: Aboit the deterministic simulation of nondeterministic (log n)-tape bounded Turing machines, 2-te TI Fachtagung Automatentheorie und Formale Sprachen, 1975, Springer LNCS **33**, 118 - 126.

[48] B. Monien, I. Sudborough: The interface teween language theory and complexity theory, in "Formal Languages - Perspectives and Open Problems" edited by R. Book, Academic Press, New York, 1980, 287 - 324.

[49] N. Pippenger: On simultaneous resource bounds, Proc. of 20th FOCS (1979), 307 - 311.

[50] G. Rozenberg: On a family of acceptors for some classes of developmental languages, Int. J. Comput. Math. **4** (1974), 199 - 228.

[51] G. Rozenberg, A. Salomaa: The Mathematical Theory of L Systems, Academic Press, New York, 1980.

[52] W. Ruzzo, J. Simon, M. Tompa: Space-Bounded hierarchies and probabilistic computations, J. Comput. System Sci. **28** (1984), 216 - 230.

[53] W. Rytter: 'Parallel time. O(LOG N) recognition of unambiguous context-free languages', Information and Computation **73** (1987), 75 - 86.

[54] W. Savitch: Relationships between nondeterministic and deterministic tape complexities, J. Comput. System Sci. **4** (1970), 177 - 192.

[55] U. Schöning, K. Wagner: Collapsing oracle hierarchies, cesnsus functions and logarithmically many queries, Proc. of 5th STACS (1988), Springer LNCS **294**, 91 - 97.

[56] E. Shamir, C. Beeri: Checking stacks and context-free programmed grammars accept p-compete languages, Proc. of the 2nd International Colloquium on Automata, Languages, and Programming, 1974, Springer LNCS **14**, 277 - 283.

[57] L. Stockmeyer, A. Meyer: Word problems requiring exponential time: preliminary report, Proc. of the 5th Annual ACM Symposium on Theory of Computing, 1973, 1 - 9.

[58] L. Stockmeyer, C. Vishkin: Simulation of random access machines by circuits, Siam J. on computing **13** (1984), 409 - 422.

[59] I. Sudborough: A note on tape-bounded complexity classes and linear context-free languages, J. Assoc. Comput. Mach. **22** (1975), 499 - 500.

[60] I. Sudborough: The complexity of the membership problem for some extensions of context-free languages, Internat. J. Comput. Math. SECT A **6** (1977), 191 - 215.

[61] I. Sudborough: On the tape complexity of deterministic context-free languages, J. Assoc. Comput. Mach. **25** (1978), 405 - 414.

[62] R. Sczelepcsényi: The method of forcing for nondeterministic automata, EATCS-Bull. **33** (1987), 96 - 100.

[63] J. van Leeuwen: The membership question for ETOL-languages is polynomially complete, Inform. Process. Lett. **3** (1975), 138 - 143.

[64] J. van Leeuwen: Variations of a new machine model, Proc. of 17th FOCS (1976), 228 - 235.

[65] H. Venkateswaran: Properties that characterize LOGCFL, Proc. of 19th STOC (1987), 141 - 150.

[66] K. Wagner, G. Wechsung: Computational complexity, VEB Deutscher Verlag der Wissenschaften, Berlin, 1986.

[67] M. Warmuth, D. Haussler: On the complexity of iterated shuffle, J. Comp. Systems Sci. **28** (1984), 345 - 358.

[68] C. Wrathall: Complete sets and the polynomial hierarchy, Theoret. Comput. Sci. **3** (1976), 23 - 33.

[69] D. Younger: Recognition and parsing of context-free languages in time N^3, Information and Control **10** (1987), 189 - 208.

IMYCS'88, Smolenice Castle, November 14-18, 1988.

COMMUTATIONS and LANGUAGE FAMILIES†

Michel Latteux
L.I.F.L., University of Lille Flandres Artois.

ABSTRACT *This survey presents some results concerning total commutations, partial commutations and semi-commutations in connection with the families of rational and algebraic languages and more generaly with (faithful) rational cones..*

INTRODUCTION

The study of free partially commutative monoids has been initiated by Cartier and Foata [11] .It happened that words on partially commutative alphabets became of interest to computer scientists studying problems of concurrency control.Thus, traces languages, which are subsets of a free partially commutative monoid, have been proposed by Mazurkiewicz [44] as a tool for describing the behaviour of concurrent program schemes. Recently several surveys deal with partially commutative monoids (Aalbersberg and Rozenberg [1], Berstel and Sakarovitch [7], Cori [19], Mazurkiewicz [45], Perrin [51]).Other results on this subject can be found in [21,22,23,24,48] .We shall focus here on (partial) commutations which are unary operations on languages associated with partially commutative alphabets . We shall examine also the semi-commutation operation which is a useful and natural extension of the partial commutation operation .

A partially commutative alphabet is a couple (A,θ) where A is an alphabet and θ, the commutation relation, is a symetric and irreflexive binary relation over A. Associated to the commutation relation θ, one can define a mapping $f_\theta: 2^{A^*} \rightarrow 2^{A^*}$ by : for every language L over the alphabet A, $f_\theta(L)$ is the set of words equivalent to some word in L for the congruence generated by θ Thus f_θ is a unary operation on languages, called the partial commutation associated to θ. It seems useful to study this operation in connection with classical operations such as (faithful) rational transductions, shuffle and intersection(see [6,28] for precise definitions of these operations).

† This work has been partially supported by the Programme de Recherche Coordonnée "Mathématiques et Informatique" du Ministère de la Recherche et de la Technologie.

Since partial commutations generalize total commutations, we shall start this study by recalling in section 1 the main results concerning total commutations. For each of these results it is quite natural to wonder whether it remains true for partial commutations. On one hand, this can raise new questions, on the other hand this points out important differences between partial and total commutations. The results presented in this section concern mainly rational and algebraic (or context-free) commutative languages, rational cones generated by commutative languages and commutatively closed rational cones.

Section 2 is devoted to semi-commutations. If θ is an irreflexive binary relation over an alphabet A, one can consider the semi-Thue system whose the set of rules is $\{yx \rightarrow xy/(y,x) \in \theta\}$. For $L \subset A^*$, $f_\theta(L)$ denotes the set of words which can be derived in the semi-Thue system from some word of L. For instance, if $A=\{a,b\}$, $\theta=\{(b,a)\}$ and $L=(ab)^*$, then $f_\theta(L)$ is the semi-Dyck language over one pair of parentheses, a language which often appears for describing synchronization processes. It is important to note that here for u in A^*, $f_\theta(u)$ is not an equivalent class. However, several results proved originally for partial commutations remain true for semi-commutations.

In section 3 we shall present the main results on partial commutations which cannot be extended to semi-commutations . In particular, two nice characterizations of the family of rational languages closed under θ-commutations will be enonced. The first one, due to Ochmanski [49] uses operations which preserve this family. The second one, due to Zielonka [61] deals with asynchronous automata, a new type of finite automaton.

1. TOTAL COMMUTATIONS

We shall first examine commutative languages, that is, languages closed under total commutations. Commutative rational languages admit simple characterizations. In particular rational unary languages have been studied by Salomaa [58] :

Proposition *Let R be a language over the alphabet A={a}.Then the following properties are equivalent:*

i) R is rational,

ii) R is a finite union of languages of the form xy with x,y in a*,*

iii)R=F $\cup$ Gz where z is in a* and F,G are finite subsets of a*.*

Let us consider now the alphabet $A=\{a_1, a_2, \ldots, a_n\}$ and a commutative rational language $R \subset A^*$. Then, $R=Com(K)$ is the commutative closure of the rational language $K=R \cap a_1^* a_2^* \ldots a_n^*$ which is a finite union of languages of the form $R_1 R_2 \ldots R_n$ where $\forall i \in \{1, \ldots, n\}$, R_i is a rational language included in a_i^*. From the above proposition, it follows that R is a finite union of languages of the form $Com(x_1 y_1^* \ldots x_n y_n^*)$ where $\forall i \in \{1, \ldots, n\}$, $x_i, y_i \in a_i^*$. But, for commutative languages L,M, $Com(LM)=L \sqcup M$, the shuffle of L and M. This implies the following characterization of commutative rational languages:

Proposition *Let R be a language over the alphabet* $A=\{a_1, a_2, .., a_n\}$. *Then R is a commutative rational language if and only if R is a finite union of languages of the form* $x_1y_1^* \sqcup .. \sqcup x_ny_n^*$ *where* $\forall i \in \{1, .., n\}$, $x_i, y_i \in a_i^*$.

More recently, a necessary and sufficient condition for the rationality of commutative languages has been established[41] :

Proposition *Let R be a commutative language over the alphabet A. Then R is rational if and only if there exists a positive integer N such that for each* $u \in R$ *and for each* $x \in A$, $|u|_x \geq N$ *implies* $u(x^N)^* \subset R$.

Let R be a rational language. By looking at its syntactic monoïd it is easy to determine whether R is a commutative language(more generally, it permits to find the largest semi-commutation θ such that $f_\theta(R)=R$).On the other hand, it is undecidable to determine whether an algebraic language is commutative[12]. More difficult is to determine whether Com(R) remains rational. However, Ginsburg and Spanier[29], and Gohon[31] have proved the decidability of this problem.In the particular case where $R=F^*$ with F a finite language over the alphabet A, one can verify that $Com(F^*)$ is rational if and only if for each $x \in A$, $F \cap x^+ \neq \emptyset$. It is an open problem to find similar conditions for partial commutations (see[56]).But in [5] , a decidable necessary and sufficient condition on F is given which is equivalent to the algebraicity of $Com(F^*)$. Conversely, it could be interesting to have a simple characterization of commutative languages of the form F^* with F a finite language. A necessary condition is that $u^{-1}F^* = (A^t)^*$ for some u in F^* and some integer t [42] ,but this condition is not sufficient.In connection with this problem, note that a language L^* is commutative if and only if $Com(L \cup L^2) \subset L^*$(more generally L^* is closed under a semi-commutation θ if and only if $f_\theta(L \cup L^2) \subset L^*$).

Let us consider now the family of algebraic languages. Parikh's theorem asserts that Com(Alg)=Com(Rat), that is, for each algebraic language L there exists a rational language R such that Com(L)=Com(R). Clearly R is not included in L but Blattner and Latteux [9] have proved that one can find a bounded language included in L with the same commutative image (see [32] for a proof of this property in a larger family):

Proposition *The algebraic languages are Parikh-bounded. That is, for every algebraic language L there exist words* $y_1, .., y_n$ *such that* $Com(L) = Com(L \cap y_1^* .. y_n^*)$.

The links between commutative and bounded languages appear also in an old conjecture known as the Fliess conjecture. A language L over the alphabet $A=\{a_1, a_2, .., a_n\}$ is said to be a B-CF language if for every permutation σ of $\{1, .., n\}$, $L \cap a_{\sigma(1)}^* a_{\sigma(2)}^* \ldots\ldots a_{\sigma(n)}^* \in$ Alg. The Fliess conjecture was: Every commutative B-CF language is algebraic (see [3,53]). This conjecture holds for three letters alphabets([43], [50], [52]) and for arbitrary alphabets when the language is of the

form Com(FP*) with F,P finite languages [5]. Recently, Kortelainen [36] has disproved this conjecture with the language :

$$K_0=Com((a_1^3a_2a_3a_4)^*) \cup \{u \in \{a_1,a_2,a_3,a_4\}^* / |u|_{a_1} \neq |u|_a+|u|_a+|u|_{a_4}\}.$$

Then $K_0 \cap (a_1a_2)^*(a_1a_3)^*(a_1a_4)^* = \{(a_1a_2)^n(a_1a_3)^n(a_1a_4)^n / n \geq 0\} \notin \mathcal{Alg}$, but it can be verified that K_0 is a commutative B-CF language. In this example $\mathcal{C}(K_0)$, the rational cone generated by K_0 contains a non algebraic bounded language. Hence it can be conjectured that every non algebraic commutative language dominates by rational transduction a non algebraic bounded language. Since $\mathcal{C}(Com)$, the rational cone generated by commutative languages contains only Parikh-bounded languages one can show that this conjecture is equivalent to:

Conjecture *A commutative language $L \subset A^*$ is algebraic if and only if $\forall y_1,..,y_n \in A^*$, $L \cap y_1^* ..y_n^*$ is algebraic.*

The best known example of algebraic commutative language is the language $D_1^* = Com((ab)^*)$. This language is not of finite index or equivalently it is not a quasirational language and it is quite natural to wonder if there exist non rational algebraic commutative languages of finite index. More generally, one can try to determine those subfamilies of Alg for which commutativity implies rationality. By using the second characterization of commutative rational languages given above and different pumping lemmas it has been shown that every commutative linear language is rational [25] and that every commutative restricted one-counter language is rational [41] .In order to prove that every algebraic commutative language of finite index is rational [35], Kortelainen has proved an important property of the family Com(Rat).This family has a minimal language for rational transduction, namely the language $\bar{D}_1^* = (a+b)^* - D_1^*$:

Proposition *If L is a non rational commutative language in $\mathcal{C}(\bar{D}_1^*)$, the rational cone generated by $\bar{D}_1^*$, then $\mathcal{C}(L) = \mathcal{C}(\bar{D}_1^*)$.*

Since the family of finite index algebraic languages is a rational cone and that the language $\bar{D}_1^*$ is not of finite index [40], it follows:

Proposition *Every algebraic commutative language of finite index is rational.*

The above results show that a lot of algebraic languages do not dominate any non rational commutative language. Conversely, it is shown in [38] that several algebraic languages do not belong to $\mathcal{C}(Com)$, the rational cone generated by the family of arbitrary commutative languages. For instance, the families of linear languages and of restricted one-counter languages are not included in $\mathcal{C}(Com)$. Thus Alg does not posses any commutative generator(a commutative language L such that $\mathcal{C}(L)=Alg$). However it could happen that there exists an algebraic commutative language which dominates every algebraic commutative language by rational transductions. Note that the language D_1^* cannot be such a

candidate since the algebraic language $Com((ab)^*(ac)^*(bc)^*) \notin \mathcal{C}(D_1^*)$ [38] and the following conjecture appears in [39]:

CONJECTURE *The rational cone generated by the family of algebraic commutative languages is not principal.*

Ginsburg and Spanier have stated in[30] the same conjecture for the rational cone generated by $Com(\mathcal{Rat})$,the family of the commutative closures of rational languages. This have been solved positively in [37] where it was proved that $\mathcal{C}(Com(\mathcal{Rat}))$ is equal to $\mathcal{C}_\cap(C_1)$, the intersection closed rational cone generated by the language $C_1=\{a^nb^n/n>0\}$. Since an intersection closed rational cone $\mathcal{L}$ is commutatively closed (i.e. $Com(\mathcal{L}) \subset \mathcal{L}$) if and only if $C_1 \in \mathcal{L}$ [37] , it follows that $\mathcal{C}(Com(\mathcal{Rat}))$ is the smallest commutatively closed rational cone. Moreover we have the equality : $\mathcal{C}(Com(\mathcal{Rat})) = \mathcal{H}_{sa}(Com(\mathcal{Rat}) \cap \mathcal{Rat})$, that is every language in $\mathcal{C}(Com(\mathcal{Rat}))$ is the image by a strictly alphabetic (i.e. length preserving) homomorphism of the intersection of a language in $Com(\mathcal{Rat})$ with a rational language. If we consider $\mathcal{Rat}_n$, the family of rational languages defined on n letters alphabets, $Com(\mathcal{Rat}_n)$ generates a principal rational cone[34]. More precisely, $Com((a_1a_2..a_n)^*)$ and $Com((a_2b_2)^*..(a_nb_n)^*)$ are two generators of $\mathcal{C}(Com(\mathcal{Rat}_n))$.

Let us consider now $\mathcal{C}(\mathcal{Com})$, the rational cone generated by $\mathcal{Com}$, the family of arbitrary commutative languages. Then $\mathcal{C}(\mathcal{Com}) = \mathcal{H}_{sa}(\mathcal{Com} \cap \mathcal{Rat}) = \mathcal{C}_\cap(\mathcal{Un})$, the intersection closed rational cone generated by $\mathcal{Un}$, the family of unary languages (languages defined on a one letter alphabet) [33]. The intersection is mandatory since every algebraic commutative language in $\mathcal{C}(\mathcal{Un})$ is rational; but $\mathcal{C}(\mathcal{Un})$ contains a non rational algebraic language [4]. Lastly, every language in $\mathcal{Com}_{re}$, the family of recursively enumerable commutative languages can be obtained from the language $SQE=\{a^nb^{n^2}/n>0\}$ by intersection and rational transductions [60] hence we get the equality $\mathcal{C}(\mathcal{Com}_{re}) = \mathcal{C}_\cap(SQE)$.

2. SEMI-COMMUTATIONS

Semi-commutations which have been introduced by Clerbout in her thesis[12](see also [15,16,59]) correspond to particular rewriting systems. In such systems, the rules are of the form ba $\rightarrow$ ab where a and b are two different letters. This implies the existence of some useful "derivation" lemmas. The first one due to Clerbout [12] states that the image of a word by a semi-commutation can be reconstructed from the images by this semi-commutation of the projections of this word on the two letters subalphabets. Let us consider a semi-commutation f_θ on the alphabet A. For a,b in A, $\pi_{a,b}$ denotes the projection on the alphabet {a,b}.

Proposition (Projection Lemma) *Let f_θ be a semi-commutation on an alphabet A. For $u,v \in A^*$, $v \in f_\theta(u)$ if and only if $\forall\ a,b \in A$, $\pi_{a,b}(v) \in f_\theta(\pi_{a,b}(u))$.*

It is often useful to consider words which do not contain two occurrences of the same letter. For that,we can use a gsm which numerotes the different

occurrences of the same letter [55]. More precisely, for every positive integer k, if we define:

- The alphabet $A_k = A \times \{1,...,k\}$,
- The semi-commutation relation $\theta_k = \{((a,i),(b,j)) \in A_k \times A_k / (a,b) \in \theta\}$,
- The gsm num_k by: $num_k(\varepsilon) = \varepsilon$ and $\forall u \in A^*$, $a \in A$, $num_k(ua) = num_k(u)(a,s)$ with $s = inf(k, |ua|_a)$,

then, we can state:

Proposition (Numerotation Lemma) *Let f_θ be a semi-commutation on an alphabet A. For $u,v \in A^*$, $v \in f_\theta(u)$ if and only if $num_k(v) \in f_{\theta_k}(num_k(u))$.*

In particular, if $k \geq |u|$, $num_k(u)$ do not contain two occurrences of the same letter. Now, by using the two previous lemmas it can be proved other useful results. For instance, in[46], it is shown that if $xy \in f_\theta(uv)$, there exist $x_1,x_2,y_1,y_2 \in A^*$ such that $x_1y_1 \in f_\theta(u)$, $x_2y_2 \in f_\theta(v)$, $x \in f_\theta(x_1x_2)$, $y \in f_\theta(y_1y_2)$ and for every letter $a \in alph(y_1)$ (the set of the letters occuring in y_1), and for every letter $b \in alph(x_2)$, $(a,b) \in \theta$. In [18], it is proved that if uv and $u'v' \in f_\theta(w)$, with $Com(u) = Com(u')$, then there exist $t,z \in A^*$ such that $tz \in f_\theta(w)$, $u,u' \in f_\theta(t)$, $v,v' \in f_\theta(z)$. Hence, uv' and u'v belong to $f_\theta(w)$.

Two subcases of semi-commutations are particularly interesting. When the relation θ is symmetric, we get partial commutations which are linked with the theory of trace languages. If moreover $\bar{\theta} = A \times A - \theta$ is an equivalence relation corresponding to a partition $\{A_1,...,A_k\}$ of A, we get partitionned commutations (see [14]). Then, for every $u \in A^*$, $f_\theta(u) = \pi_1(u) \sqcup ... \sqcup \pi_k(u)$, where π_i is the projection on the subalphabet A_i. The composition of partitionned commutations have been studied in[13,55] and several problems remain open on this subject.

For a language family $\mathcal{L}$, $SC(\mathcal{L})$ (resp. $PC(\mathcal{L})$, $P(\mathcal{L})$) denotes the family of the languages of the form $f_\theta(L)$ where $L \in \mathcal{L}$ and f_θ is a semi-commutation (resp. a partial commutation, a partitionned commutation). Then, infinite hierarchies are obtained from the family of rational languages since $\forall k \in \mathbb{N}$, $P^{k+1}(\mathcal{Rat}) - SC^k(\mathcal{Rat}) \neq \emptyset$ ($P^k(\mathcal{Rat})$ is, by definition equal to $P(P^k(\mathcal{Rat}))$ [12]. Moreover, every (faithful) rational cone closed under partitionned commutations is closed under semi-commutations[12] and an intersection closed (faithful) rational cone is closed under partitionned commutations if and only if it contains the language $Copy = \{uu / u \in (a+b)^*\}$[14]. Since it is easy to verify that $\mathcal{C}^f(P(\mathcal{Rat}))$ is closed under intersection, we get that $\mathcal{C}^f_\cap(Copy)$, known as the family MULTI-RESET[10], is the smallest faithful rational cone closed under semi-commutation. More precisely:

Proposition *The family MULTI-RESET=$\mathcal{C}^f_\cap(Copy)$ is equal to $\mathcal{H}_{sa}(P(\mathcal{Rat}) \cap \mathcal{Rat})$. Hence, MULTI-RESET is the smallest faithful rational cone closed under semi-commutation.*

Since $\mathcal{C}_\cap(Copy)$ is equal to r.e., the family of recursively enumerable languages, we can deduce that $r.e. = \mathcal{H}_a(P(\mathcal{Rat}) \cap \mathcal{Rat})$ is the smallest rational cone closed under semi-commutation [14].

Let us consider, now, θ-closed rational languages R_1, $R_2 \subset A^*$ ($f_\theta(R_i) = R_i$ for i= 1,2). Then, $f_\theta(R_1 \cup R_2) = f_\theta(R_1) \cup f_\theta(R_2) \in \mathfrak{Rat}$, and $f_\theta(R_1R_2)$ is also a rational language[16].In order to obtain a sufficient condition for the rationality of $f_\theta(R_1^*)$ we need the notion of non-commutation graph of θ, noted $(A,\bar{\theta})$. The vertices of $(A,\bar{\theta})$ are the letters of A and (b,a) is an edge iff $(b,a) \in \bar{\theta}$. If, for every u in R_1, $(alph(u), \bar{\theta})$, the partial subgraph of $(A,\bar{\theta})$, corresponding to the letters occurring in the word u, is strongly connected, then $f_\theta(R_1^*)$ is a rational language [16]. More generaly,Metivier [46] has established the following result:

Proposition *Let $R \subset A^*$ be a rational language such that for every iterating factor u of R, the partial subgraph $(alph(u), \bar{\theta})$ is strongly connected. Then $f_\theta(R)$ is a rational language.*

The non-commutation graph of θ permits also to characterize the words u such that $f_\theta(u^*)$ is an algebraic language [18]:

Proposition *For u in A^* the language $f_\theta(u^*)$ is algebraic if and only if the partial subgraph $(alph(u), \bar{\theta})$ has at most two strongly connected components.*

Of particular interest are the algebrico-rational semi-commutations which, by definition, transform rational languages in algebraic languages. These semi-commutations admit the following characterization [17]:

Proposition *The semi-commutation f_θ is algebrico-rational if and only if for every $(a,b) \in \theta$ there do not exist $c,d \in A-\{a,b\}$ such that (a,c) and $(d,b) \in \theta$.*

3. PARTIAL COMMUTATIONS

In this section, we shall mention several important results concerning partial commutations which do not exist for semi-commutations. Firstly, the "embedding theorem" due to Clerbout and Latteux [14](see also [51]) shows that partitionned commutations play a central role for studying partial commutations:

Proposition *Let f_θ be a partial commutation over the alphabet A. Then, there exist a partitionned commutation $f_{\theta'}$ over the alphabet $B = A \times A$ and a morphism $g: A^* \to B^*$ such that: $\forall u,v \in A^*$, $v \in f_\theta(u)$ if and only if $g(v) \in f_{\theta'}(g(u))$.*

Let $a_1, a_2, \ldots, a_n$ be an ordering of the alphabet A. The characterization of commutative rational languages given at the first section implies that a commutative language L is rational if and only if $L \cap a_1^* a_2^* \ldots a_n^*$ is rational. The language $K = a_1^* a_2^* \ldots a_n^*$ is the set of minimal words for the total commutation in the sense that a word u is in K if u is the smallest element of Com(u) for the lexicographical ordering of A^*. Now, if f_θ is a partial commutation, one says that a word u is θ-minimal if it is the smallest element of $f_\theta(u)$ for the lexicographical ordering of A^*. Then $Min(A^*)$ denotes the set of θ-minimal words. Clearly, $Min(A^*)$ is a rational language since its complement is equal to the union for all $(a_i,a_j) \in \theta$

and i<j of the languages $A^*a_jA_i^*a_iA^*$ where $A_i = \{ a_s \in A / (a_i,a_s) \in \theta\}$. Then, for every rational language $R \subset A^*$, $Min(R) = Min(A^*) \cap R$ is rational. Ochmanski has proved in [49] that the converse is true for θ-closed languages:

Proposition *Let f_θ be a partial commutation over the alphabet A and $R \subset A^*$ a θ-closed language . Then R is rational if and only if Min(R) is rational.*

This proposition is useful for the proof of the characterization theorem of Ochmanski[49] which asserts that the family of θ-closed rational languages over the alphabet A is the closure of the family of elementary languages (languages included in $A \cup \{\varepsilon\}$) under some operations on languages. These operations are the union,the θ-product which, for $R,R' \subset A^*$ gives $f_\theta(RR')$ and a new operation, the asynchronous star operation which is defined below. For u in A^*, let us take $A_1,\ldots,A_s$, the maximal subalphabets of alph(u) such that the graphs $(A_i,\bar{\theta})$ are strongly connected. Then AS(u) denotes the set $\{\pi_1(u),\ldots,\pi_s(u)\}$ where for i in $\{1,\ldots,s\}$, $\pi_i(u)$ is the projection of u on the alphabet A_i. For $R \subset A^*$, $AS(R) = \{v \in AS(u) / u \in R \}$ and the asynchronous star of R is equal to $f_\theta(AS(R)^*)$.

Proposition *The family of θ-closed rational languages is the smallest family of languages containing the family of elementary languages and closed under union, θ-product and the asynchronous star operation.*

Now, if we consider the family $f_\theta(\mathfrak{Rat}(A^*))$, that is $\{f_\theta(R) / R \text{ rational}; R \subset A^*\}$. This family is closed under union but, generally, it is not closed under complement. However, if θ is the total commutation, $f_\theta(\mathfrak{Rat}(A^*))=Com(\mathfrak{Rat}(A^*))$ is closed under the boolean operations [26,27]. Partial commutations for which this property holds have been characterized in [2,8,57]:

Proposition *Let f_θ be a partial commutation over the alphabet A. Then $f_\theta(\mathfrak{Rat}(A^*))$ is closed under the boolean operations if and only if $\theta \cup \{(a,a) / a \in A\}$ is an equivalence relation.*

An important characterization of θ-closed rational languages has been given by Zielonka in terms of asynchronous automata, a special type of finite deterministic automaton[61]. A θ-asynchronous automaton is such that if $(a,b) \in \theta$, then for every state q, the state reached by reading ab from q is necessarily the same that the state reached by reading ba from q. Hence, the language recognized by a θ-asynchronous automaton is surely a θ-closed rational language. Roughly, in such an automaton, the states are n-tuples, each letter of the alphabet A can only rise to a given subset of the components of the states and two permutting letters cannot rise to the same component. Let us give a precise definition of this type of automaton:

A *θ-asynchronous automaton* is a couple $(\mathcal{A}, \gamma)$, where $\mathcal{A} = (A,Q,q_0,*,F)$ is a finite deterministic automaton with $Q = Q_1x\ldots xQ_n$ and γ is a mapping from A to the family of subsets of $\{1,\ldots,n\}$ such that:

-θ is equal to $\{(a,b) \in AxA \ / \ \gamma(a) \cap \gamma(b) = \emptyset\}$,

-$\forall\ a \in A$, $q \in Q$ and $i \notin \gamma(a)$, the i^{th} component of $q*a$, noted $[q*a]_i$, is equal to $[q]_i$,

-$\forall\ a \in A$, $q, q' \in Q$, if $\forall\ i \in \gamma(a)$, $[q]_i = [q']_i$, then $\forall\ i \in \gamma(a)$, $[q*a]_i = [q'*a]_i$.

The difficult part of the theorem of Zielonka consists in defining a θ-asynchronous automaton for a given θ-closed rational languages:

Proposition *The family of θ-closed rational languages is equal to the family of languages recognized by θ-asynchronous automata.*

Since, several attempts have been made to give simpler proofs for this result[47,56]. The main lesson of these attempts is to show the intrinsic difficulty of this theorem. In [20], it is slightly extended. Indeed, it is proved that every θ-closed rational language can be recognized by a θ-asynchronous automaton $(\mathcal{A}, \gamma)$ in which $\gamma^{-1}(i)$ contains at most two elements. At last, the notion of virtually asynchronous automaton is introduced in [54]: a finite deterministic automaton is virtually θ-asynchronous if it can be transformed into a θ-asynchronous one only by renaming its states. A simple algorithm is then given to determine whether a finite deterministic automaton is virtually θ-asynchronous and if the answer is positive, the corresponding θ-asynchronous automaton is easily built.

REFERENCES

[1] IJ.J.Aalbersberg and G.Rozenberg, Theory of traces, Tech.Rep., University of Leiden, 1986.

[2] IJ.J.Aalbersberg and E.Welzl, Trace languages defined by regular string languages, RAIRO Inform.Theor. **20**(1986) 103-119.

[3] J.M.Autebert, J.Beauquier, L.Boasson and M.Latteux, Very small families of languages, in R.V.Book,ed., Formal Language Theory, Perspective and Open Problems (Academic Press, New York,1980) 89-107.

[4] J.M.Autebert, J.Beauquier, L.Boasson and M.Latteux, Langages algébriques dominés par des langages unaires, Information and Control **48**(1981) 49-53.

[5] J.Beauquier, M.Blattner and M.Latteux, On commutative context-free languages, J.of Comput. and Syst.Sc. **35**(1987).

[6] J.Berstel,Transductions and Context-Free Languages (Teubner,1979).

[7] J.Berstel and J.Sakarovich, Recent results in the theory of rational sets, Lect.Notes in Comp.Sci. **233**(1986) 15-28.

[8] A.Bertoni, G.Mauri and N.Sabadini, Unambiguous regular trace languages, in J.Demetrovics, G.Katona and A.Salomaa, eds., Algebra, Combinatorics and Logic in Computer Science (North Holland, 1985).

[9] M.Blattner and M.Latteux, Parikh-bounded languages, Lect.Notes in Comp.Sci. **115**(1981) 316-323.

[10] R.V.Book, S.Greibach and C.Wrathall, Reset machines, J.of Comput. and Syst.Sc. **19**(1979) 256-276.

[11] P.Cartier and D.Foata, Problèmes combinatoires de commutations et réarrangements, Lect. Notes in Math. **85**(1969).

[12] M.Clerbout, Commutations partielles et familles de langages, Thesis, University of Lille, 1984.

[13] M.Clerbout, Compositions de fonctions de commutation partielle, to appear in RAIRO Inform.Theor. , 1986.

[14] M.Clerbout and M.Latteux, Partial commutations and faithful rational transductions, Theoretical Computer Science **34**(1984) 241-254.

[15] M.Clerbout and M.Latteux, On a generalization of partial commutations, in: M.Arato, I.Katai, L.Varga, eds, Proc.Fourth Hung. Computer Sci.Conf. (1985) 15-24.

[16 M.Clerbout and M.Latteux, Semi-commutations, Information and Computation **73**(1987) 59-74.

[17] M.Clerbout and Y.Roos, Semi-commutations algebrico-rationnelles, Tech. Rep. n^0 126-88, University of Lille, 1988.

[18] M.Clerbout and Y.Roos, Semi-commutations et langages algébriques, Tech.Rep. n^0 129-88, University of Lille, 1988.

[19] R.Cori, Partially abelian monoids, Invited lecture, STACS, Orsay, 1986.

[20] R.Cori, M.Latteux, Y.Roos and E.Sopena, 2-asynchronous automata, to appear in Theoretical Computer Science, 1987.

[21] R.Cori and Y.Metivier, Recognizable subsets of partially abelian monoids, Theoretical Computer Science **38**(1985) 179-189.

[22] R.Cori and D.Perrin, Automates et commutations partielles, RAIRO Inform.Theor. **19**(1985) 21-32.

[23] C.Duboc, Some properties of commutation in free partially commutative monoids, Inform.Proc.Letters **20**(1985) 1-4.

[24] C.Duboc, Commutations dans les monoïdes libres: un cadre théorique pour l'étude du parallélisme, Thesis, University of Rouen, 1986.

[25] A.Ehrenfeucht, D.Haussler and G.Rozenberg, Conditions enforcing regularity of context-free languages, Lect.Notes in Comp.Sci. **140**(1982) 187-191.

[26] S.Eilenberg and M.P.Schützenberger, Rational sets in commutative monoids, J. of Algebra **13**(1969) 344-353.

[27] S.Ginsburg, The Mathematical Theory of Context-Free Languages (McGraw-Hill, New York,1966).

[28] S.Ginsburg, Algebraic and Automata-Theoretic Properties of Formal Languages (North Holland, Amsterdam, 1975).

[29] S.Ginsburg and E.H.Spanier, Semigroups, Preburger formulas and languages, Pacif.J.Math. **16**(1966) 285-296.

[30] S.Ginsburg and E.H.Spanier, AFL with the semilinear property, J.of Comput. and Syst.Sc. **5**(1971) 365-396.

[31] Ph.Gohon, An algorithm to decide whether a rational subset of $\mathbb{N}^k$ is recognizable, Theoretical Computer Science **41**(1985) 51-59.

[32] A.K.Joshi and T.Yokomori, Semi-linearity,Parikh-boundedness and tree adjunct languages, Inform.Proc.Letters **17**(1983) 137-143.

[33] J. Kortelainen, On language families generated by commutative languages, Ph.D. Thesis, University of Oulu, 1982.

[34] J. Kortelainen, A result concerning the trio generated by commutative slip-languages, Discrete Applied Mathematics **4**(1982) 233_236.

[35] J. Kortelainen, Every commutative quasirational language is regular, RAIRO Inform.Theor. **20**(1986) 319-337.

[36] J.Kortelainen, The conjecture of Fliess on commutative context-free languages,to appear, 1988.

[37] M. Latteux, Cones rationnels commutativement clos, RAIRO Inform.Theor.**11**(1977) 29-51.

[38] M. Latteux, Cones rationnels commutatifs, J.of Comput. and Syst.Sc. **18**(1979) 307_333.

[39] M. Latteux, Langages commutatifs, transductions rationnelles et intersection, in M.Blab ed., Actes de l'école de printemps de théorie des langages (Tech.Rep.82-14, LITP, 1982) 235-242.

[40] M. Latteux and J.Leguy, On the usefulness of bifaifhful rational cones, Math.Systems Theory **18**(1985) 19-32.

[41] M. Latteux and G. Rozenberg, Commutative one-couter languages are regular, J.of Comput. and Syst.Sc. **29**(1984) 54_57.

[42] M.Latteux and G.Thierrin, Codes and commutative star languages, Soochow J.of Math. **10**(1984) 61-71.

[43] H.A.Maurer, The solution of a problem by Ginsburg, Inform.Process.Lett. **1**(1971) 7-10.

[44] A.Mazurkiewicz, Concurrent program schemes and their interpretations, DAIMI PB 78, University of Aarhus, 1977.

[45] A.Mazurkiewicz, Traces, histories and graphs: instances of process monoids, Lect.Notes in Comp.Sci. **176**(1984) 115-133.

[46] Y.Metivier, Semi commutations dans le monoïde libre,Tech. Rep. n^0 I-8606, University of Bordeaux, 1986.

[47] Y.Metivier, Contribution à l'étude des monoïdes de commutations,Thèse d'état,University of Bordeaux, 1987.

[48] Y.Metivier, On recognizable subsets of free partially commutative monoids, Lect.Notes in Comp.Sci. **226**(1986) 254-264.

[49] E.Ochmanski, Regular behaviour of concurrent systems, Bulletin of EATCS **27**(1985) 56_67.

[50] T.Oshiba,On permutting letters of words in context-free languages, Information and Control **20**(1972) 405-409.

[51] D.Perrin, Words over a partially commutative alphabet, NATO ASI Series F12,Springer (1985) 329-340.

[52] J.F.Perrot, Sur la fermeture commutative des C-langages, C.R.Acad.Sci.Paris **265**(1967) 597-600.

[53] A.Restivo and C.Reutenauer, Rational languages and the Burnside problem, Theoretical Computer Science **40**(1985) 13-30.

[54] Y.Roos, Virtually asynchronous automata, Conference on Automata, Languages and Programming Systems, Salgotarjan, 1988.

[55] Y.Roos, Contribution à l'étude des fonctions de commutation partielle, Thesis, University of Lille , 1988.

[56] B.Rozoy, Un modèle de parallélisme: le monoïde distribué, Thèse d'état,University of Caen, 1987.

[57] J.Sakarovitch, On regular trace languages, to appear in RAIRO Inform.Theor. , 1986.

[58] A.Salomaa, Theory of Automata, (Pergamon Press, Oxford, 1969).

[59] M.Szijarto, The closure of languages on a binary relation, IMYCS Conference, Smolenice,1982.

[60] P.Turakainen, On some bounded semiAFLs and AFLs, Inform.Sci. **23**(1981) 31-48.

[61] W. Zielonka, Notes on asynchronous automata, RAIRO Inform.Theor. **21**(1987) 99-135.

IMYCS'88, Smolenice Castle, November 14-18, 1988.

The Riches of Rectangles

Derick Wood

Data Structuring Group
Department of Computer Science
University of Waterloo
WATERLOO, Ontario N2L 3G1
CANADA

Abstract

In this paper we consider some of the rectangle problems that have been studied in the literature of computational geometry. Our aim is to demonstrate that although rectangles are, perhaps, the simplest of geometrical figures, they occur naturally in many situations and, thus they are a rich source for intriguing and challenging problems.

1 Introduction

A title such as "The Riches as Rectangles" suggests that there may be sequels such as "The Treasures of Triangles", "The Quandries of Quadrilaterals", and "The Horrors of Octagons" to name but a few. I believe this to be the case for some of these topics, since like rectangles some of them occur profusely in the real world. But, if this is the case, why have I chosen rectangles first?

I have two responses to this question. First, I have studied many rectangle problems over the last ten years and, therefore, claim some expertise with them. And, second, a rectangle is the simplest polygon apart from the square, which is a special case of a rectangle. Why is this so? Because rectangles have four sides, but these sides have only two orientations, and their angles are all right angles. (A triangle has fewer sides, but it has more orientations and more angles.)

We consider some problems for rectangles that have been studied in computational geometry and some that have not. Our hope is that the reader will obtain a little of the flavor of the types of questions, the types of results, and the types of approaches that are to be found in this rapidly growing area. In other words, we use rectangles to provide the reader with the flavor of computational geometry. [1]

[1]The term computational geometry has been in use in the areas of computer graphics and solid modeling for many years with a more restrictive meaning. We are interested in computational aspects of combinatorial geometry problems here.

2 What Are Rectangles

We assume that rectangles are given by quadruples of the form $(x_{left}, x_{right}, y_{bottom}, y_{top})$ that determine their corner points. Hence, we assume they have sides parallel to the two axes in two-dimensional space. For the purposes of this paper we assume that all coordinate values are integral. A rectangle with sides parallel to the axes is termed *isothetic* or *orthogonal.* Many problems involve isothetic sets of rectangles.

Given two rectangles $R^i = (x_l^i, x_r^i, y_b^i, y_t^i)$, $i = 1, 2$, we say that R^1 is a *zoom* or *homothet* of R^2 if $(x_r^1 - x_l^1)(y_t^2 - y_b^2) = (x_r^2 - x_l^2)(y_t^1 - y_b^1)$, that is, one is a magnification of the other. It is clear that every square is a zoom of every other square, but this is not true for rectangles. Therefore, for rectangles we may wish to restrict our attention to those rectangles in a homothetic class generated by some given rectangle. The most restricted class of rectangles we might consider are those consisting of a single rectangle or a single *tile.* In this case, we consider only *translates* of such a set.

Throughout this paper, as is usual, we use the term to mean either the closed set of points it defines or its boundary alone; the meaning will always be clear from the context.

3 The Combinatorial-Computational Relationship

We begin by exploring the differences between the combinatorial approach and the computational approach to geometric problems. Essentially, combinatorics characterizes when a property occurs, whereas computation computes when a property occurs, almost occurs, or doesn't occur.

As our first example, consider the following theorem. (Note that we assume all rectangles are isothetic throughout the remainder of this paper.)

Theorem 3.1 *Given n rectangles in the plane, they share a common point (or have a nonempty intersection) if and only if every pair share a common point (or intersect).*

Computationally, we might turn this into:

Problem 3.1 *Given n rectangles in the plane, do they share a common point? How fast can this be determined?*

If the rectangles have an empty intersection as the ones in Figure 1 do it is unclear how to proceed combinatorially, but computationally we can easily weaken the question.

Problem 3.2 *Given n rectangles in the plane, what is the maximum number of them that share a common point?*

This value is called the *thickness* of the rectangles; in Figure 1 the thickness is three. If we can solve the thickness problem efficiently, it should be clear that we can solve the common point problem as efficiently. This follows by observing that the n rectangles share a common point if and only if their thickness is n.

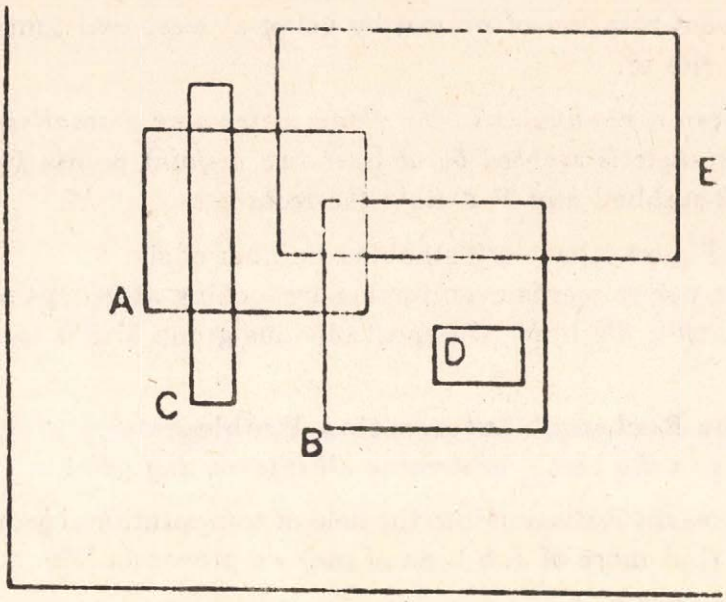


Figure 1: Some orthogonal rectangles.

A different approach is found by considering the common point (when it exists) rather than the rectangles that share it. Given n rectangles and a point p, all rectangles containing p are said to be *stabbed* by p. This gives rise to:

Problem 3.3 *Given n rectangles in the plane and a point p, which rectangles does p stab?*

This is an example of a *searching* problem. Also, this particular problem is a fundamental one, so we return to it in Section 4.

In order to consider a second weakened version of Problem 3.1, we first introduce an intermediate problem.

Problem 3.4 *Given n rectangles in the plane and a set P of points, is every rectangle stabbed by at least one point in P? (We say that P* stabs *the rectangles if this is the case.)*

Now we turn to the second weakened version of Problem 3.1.

Problem 3.5 *Given n rectangles in the plane, determine a smallest set P of points that stabs the rectangles.*

To see that this is a weakened version of Problem 3.1 observe that the n rectangles share a common point or are pairwise disjoint if and only if the smallest set P of points that stabs them has size 1 or n, respectively. We say that $\#P$ is the *stabbing number* of the given rectangles. The rectangles in Figure 1 have stabbing number three. The problem can also be viewed as a thumbtack problem. Given n notices, what is the fewest thumbtacks needed to attach them to a noticeboard with a particular overlap

pattern. Of course, in practice, the notices might rotate and we would like to prevent this. We can prevent rotation of notices by using at least two thumbtacks for each notice. This gives rise to:

Problem 3.6 *Given n rectangles in the plane, determine a smallest set P of points such that each rectangle is stabbed by at least two disjoint points from P. (We say each rectangle is* 2-stabbed *and P 2-stabs the rectangles.*

The rectangles in Figure 1 have a 2-stabbing number of six.

We can weaken our concerns even further by looking at groups of fixed size that share a common point. Perhaps, the most obvious group size is two, in which case we have:

Problem 3.7 The Rectangle Intersection Problem
Given n rectangles in the plane, determine all intersecting pairs.

This problem was my initiation into the field of computational geometry. Together with Jon Bentley (but more of Jon than of me) we proved in [1]:

Theorem 3.2 *The R.I.P. for n rectangles can be solved in $O(a + n \log n)$ time and $O(n \log n)$ space, where a is the number of answers or intersecting pairs.*

If $k = O(n^2)$, this algorithm outperforms the naive algorithm that examines the $\binom{n}{2}$ pairs. Our algorithm was subsequently improved by Edelsbrunner and McCreight independently. They both showed that $O(n)$ space to be sufficient (see [4,5,8]), but McCreight's solution did not have a provable worst case time bound of $O(a+n \log n)$. Later, McCreight gave a second alternative approach that also met these optimal time and space bounds; see [9].

The interesting point about these algorithms is that they introduced novel data structures to the computer science community — the *segment tree*, the *range tree*, the *interval tree*, and the *priority search tree* — that have become basic data structures in many other computational geometry algorithms.

The original motivation for the R.I.P. was from the area of VLSI design checking. in this setting, components, wires, and input-output pads could, at that time, be viewed as rectangles. Hence, between different layers of the chip some intersections were essential, while others were design faults. Finding all pairwise intersections so that these could be examined in more detail was one of the design checking steps.

Corresponding to the n rectangles in the plane we have their *rectangle intersection graph* in which rectangles are nodes and two nodes are incident to the same edge if the associated rectangles intersect. In Figure 2 we display the rectangle intersection graph of the rectangles in Figure 1. The R.I.P. is equivalent to determining the rectangle intersection graph of the given rectangles. This simple notion leads to a nontrivial combinatorial problem:

Problem 3.8 *Characterize the rectangle intersection graphs.*

The one-dimensional version of this problem, for line segments, has been well studied. In this case we obtain *interval graphs* which have many different characterizations; see [12].

The graph-theoretic view of rectangles leads naturally to:

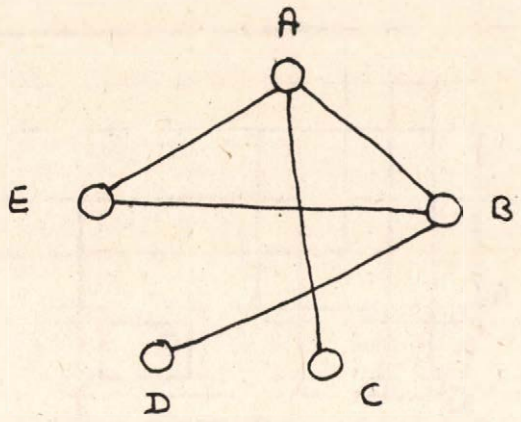


Figure 2: A rectangle intersection graph.

Problem 3.9 *Given n rectangles in the plane, determine their connected components (in the rectangle intersection graph sense).*

This is yet another important problem in VLSI design. It can be solved using the standard algorithms of graph theory. This leads to solutions that require $O(a+n)$ time, where a is the number of pairwise intersections. When $a = \omega(n \log n)$, this solution technique can be improved substantially. The basic idea is to avoid the explicit construction of the intersection graph. Edelsbrunner et al. [6] proved:

Theorem 3.3 *Given n rectangles in the plane, their connected components can be computed in $O(n \log n)$ time and $O(n)$ space.*

4 Stabbing Search

Recall that in this problem we are given n rectangles and a query point. We wish to determine all rectangles that the query point stabs.

Clearly, if this is all we are given we cannot do better than exhaustive search — test each rectangle to see if it is stabbed by the query point. However, usually we expect many query points, not just one. In this case, we can do better by *preconditioning* or *preprocessing* the rectangles in some way. For example, extend all horizontal edges of the rectangles to partition the plane into at most $2n+1$ horizontal slabs; see Figure 3. Each horizontal slab is immediately partitioned into at most $2n+1$ cells by the rectangle's vertical edges and each rectangle has been partitioned into a number of cells that only intersect at their boundaries. Therefore, we associate with each cell, in each horizontal slab, the "names" of the rectangles that they belong to.

Given a query point $p = (x, y)$, we can perform binary search with y to determine the slab in which it lies. (Assuming that we keep the y-coordinates of the slabs in sorted order in an array.) Similarly, once we know the slab that contains y, we can perform a binary search with x within the slab to determine the cell that contains p.

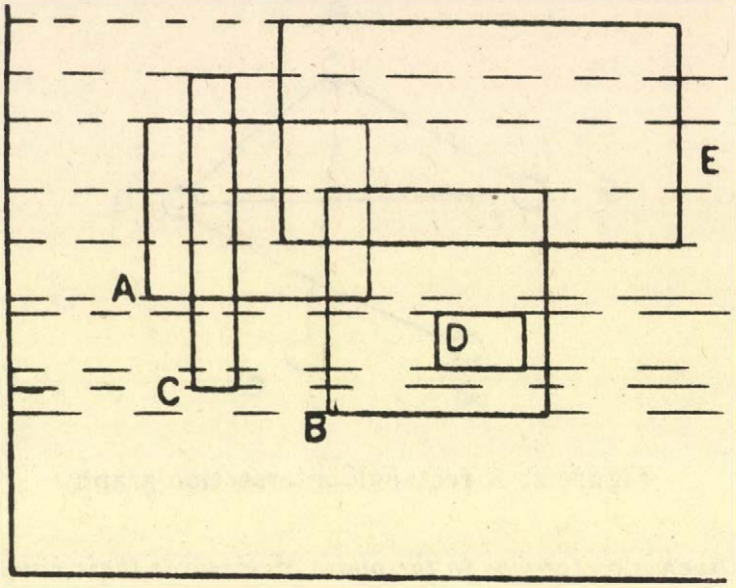


Figure 3: The slabs induced by some rectangles.

This slab technique provides $O(a + \log n)$ query time, where a is the number of answers. Unfortunately, it requires $O(n^3)$ space in the worst case and, therefore, $O(n^3)$ time to preprocess the rectangles. The reader is invited to provide n rectangles that achieve this bound.

Can we do better? Fortunately, yes we can. Observe that an orthogonal rectangle can be defined as the cross-product of the two intervals given by its x- and y-projections. Moreover, a point $p = (x, y)$ is in a rectangle R if and only if x is in its x-projection and y is in its y-projection. This suggests an approach, similar to the slab technique, except that we first perform a binary search on the y-projections of the rectangles. Second, we perform a binary search on the x-projections of only those rectangles that were not excluded in the first search. The resulting rectangles must be stabbed by p and none of the excluded rectangles are. Using tree data structures first proposed by Edelsbrunner, Six, and Wood [4,5,15] the rectangles can be preprocessed in $O(n \log^2 n)$ time to give a structure that requires $O(n \log n)$ space. Furthermore, a stabbing query can be solved in $O(a + \log^2 n)$ time. This structure is, essentially, a search tree for y-intervals and each node is a second-level search tree for x-intervals.

It is possible to do even better than this by once more changing our view of the problem. Consider the simpler stabbing problem in which we are only given y-intervals on the y-axis, rather than rectangles placed arbitrarily. Then, this problem can be solved with $O(n \log n)$ preprocessing time and $O(n)$ space using the structure of either Edelsbrunner [3] or McCreight [9]. In both cases, the query time is $O(a + \log n)$; the best we can hope for in the worst case with a comparison-based model of computation.

But now consider a modification of this problem in which y-intervals are inserted

and deleted over time and we want to ask *historical stabbing queries.* That is, we give a y-value and a time t and we want all y-intervals that exist at time t and are stabbed by y. A little thought shows that the two-dimensional space-time chart for the y-intervals consists of a set of rectangles. In this framework a rectangle denotes the existence of a y-interval for a period of time and the historical stabbing query is nothing more nor less than a stabbing query with the point (t, y). Based on this idea, Sarnak and Tarjan [13] used *persistent search structures* to solve the stabbing query problem efficiently.

Yet one more approach to the problem under discussion is based on another simple observation. The boundaries of the rectangles partition the plane into subdivisions. In this setting the stabbing query becomes: In which subdivision does the given point lie? This is a basic problem that has been studied intensely since the dawn of computational geometry and is usually known as the point location (in a planar subdivision) problem.

The interested reader should consult the texts by Preparata and Shamos [11] and Edelsbrunner [2] and the paper of Sarnak and Tarjan [13] and the references therein.

All of the above mentioned material is concerned with worst case and amortized case behavior. Thus, it leaves open the question of whether there are solutions that do well on average and are easy to implement in practical situations. We conclude this section by examining three variants of the general stabbing problem.

The first variant occurs when multiple windows are allowed in a work station environment.

Problem 4.1 The Window Selection Problem
Given n rectangles (or windows) and a point p, determine the closest rectangle that p stabs.

Think of the rectangles as carpets on the floor and the query point as someone standing on them, then the answer to the query will be the carpet that the person is touching (We assume point feet.) Or, alternately, the windows are hanging in three-space rather than in the plane and the answer to the query is the closest window to the observer that is stabbed by the point. Based on this interpretation we can use our first approach where we also keep a sorted list of rectangles associated with each cell. The second approach can also be modified to take into account the third coordinate, but this means that the preprocessing time becomes $O(n \log^3 n)$ and the query time becomes $O(a + \log^3 n)$ in the worst case. Whether the other approaches can be modified to give efficient solutions is not so clear. Also, in practice the windows are changing dynamically — shrinking, expanding, translating, being removed, and being added. We are currently reconsidering this problem, since we know of no method that is good under all circumstances.

The second variant is also motivated by a work station environment, namely, the problem of menu selection.

Problem 4.2 The Menu Selection Problem
Given n disjoint rectangles in the plane and a query point p, determine whether p stabs a rectangle and if so, which one.

The motivation for this problem is obvious; all PC's use menu selection, to some extent, as a system language; perhaps the MacIntosh is the most thorough example. Menus are rectangles and a stabbing query is a cursor.

Since the x- and y-projections of the rectangles result in $O(n)$ intervals, the first and second approaches are no better than before, except that the number of answers does not appear in the time bounds. But we, typically, cannot afford the space overhead. Field [7] gave a solution using hashing that has $O(1)$ expected query time and requires $O(n)$ space.

The third variant is found in one approach to querying in a geographic database.

Problem 4.3 *Given n rectangles in the plane and a query rectangle, determine all rectangles the query rectangle stabs.*

This problem was motivated by range queries in a geographic database for Baden-Württemberg in West Germany. Bounding rectangles were used to surround the municipalities of this province, yielding 1211 rectangles and in a second more detailed situation 48,500 rectangles were obtained. In both cases, the query of most interest was the range query or rectangle query.

This problem is easily solved using the techniques of [3,4], but it requires $O(a + \log^3 n)$ time and $O(n \log n)$ space. In a practical environment this is unacceptable.

Recently, Six and Widmayer [14] introduced a method based on the grid file of [10] which is very promising for large databases of rectangles. However, one hopes that some further improvement is possible.

The stabbing query continues to fascinate algorithmicists, since it is an easy problem to grasp, yet good efficient solutions seem to be difficult to find.

5 Concluding Remarks

We have only touched the surface of rectangle problems and solutions; there are many more that are waiting to be explored or whose present solutions are waiting to be improved.

Acknowledgement

This work was supported under a Natural Sciences and Engineering Research Council of Canada Grant No. A-5692.

References

[1] J.L. Bentley and D. Wood. An optimal worst case algorithm for reporting intersections of rectangles. *IEEE Transactions on Computers*, EC-29:571–576, 1980.

[2] H. Edelsbrunner. *Algorithms in Combinatorial Geometry.* Springer-Verlag, New York, 1987.

[3] H. Edelsbrunner. *Dynamic Rectangle Intersection Searching.* Technical Report F 47, Institut für Informationsverarbeitung, Technische Universität Graz, 1980.

[4] H. Edelsbrunner. New approach to rectangle intersections: Part I. *International Journal of Computer Mathematics*, 13:209–219, 1983.

[5] H. Edelsbrunner. New approach to rectangle intersections: Part II. *International Journal of Computer Mathematics*, 13:221–229, 1983.

[6] H. Edelsbrunner, J. van Leeuwen, Th. Ottmann, and D. Wood. Computing the connected components of simple rectilinear geometrical objects in d-space. *RAIRO Informatique théoretique*, 18:171–183, 1984.

[7] D.E. Field. *Fast Hit Detection for Disjoint Rectangles.* Technical Report 85-53, Department of Computer Science, University of Waterloo, 1985.

[8] E.M. McCreight. *Efficient Algorithms for Enumerating Intersecting Intervals and Rectangles.* Technical Report CSL-80-9, Xerox Palo Alto Reasearch Center, 1980.

[9] E.M. McCreight. Priority search trees. *SIAM Journal on Computing*, 14:257–276, 1985.

[10] J. Nievergelt, H. Hinterberger, and K.C. Sevcik. The grid file: an adaptable, symmetric multikey file structure. *ACM Transactions on Database Systems*, 9:38–71, 1984.

[11] F.P. Preparata and M.I. Shamos. *Computational Geometry.* Springer-Verlag, New York, 1985.

[12] F.S. Roberts. *Graph Theory and Its Applications to Problems of Society.* Society for Industrial and Applied Mathematics, Philadelphia, Pennsylvania, 197?

[13] N. Sarnak and R.E. Tarjan. Planar point location using persistent search trees. *Communications of the ACM*, 29:669–679, 1986.

[14] H.-W. Six and P. Widmayer. *Spatial Searching in Geometric Databases.* Technical Report 176, Institut für Angewandte Informatik, Universität Karlsruhe, 1987.

[15] H.-W. Six and D. Wood. Counting and reporting intersections of d-ranges. *IEEE Transactions on Computers*, C-31:181–187, 1982.

COMMUNICATIONS

IMYCS'88, Smolenice Castle, November 14-18, 1988.

RELOADING AND RESTRUCTURING OF NETWORK DATA BASES.

A. BODUNOV.

Computing Center of the USSR Academy
of Sciences, Moscow, USSR.

Restructuring is an important stage in data bases exploitation. Restructuring using the reloading is one of the restructuring methods.

Furthermore, reloading in itself plays an important role in the process of data base exploitation. It helps to solve a collect garbage problem in the modified data base and allows one to transfer the data base to some other place in the external memory or from one computer to another.

The proposed method allows the reloading process and restructuring of network data bases to be automatized. The method has been realized on the family of compatible Compass DBMSs based on the CODASYL proposals [2].

The traditional approach to restructuring and reloading of network data bases

Most reseach work devoted to data bases restructuring describe a three stage restructuring process:

1. the unloading of the data base to a serial file in some standard format;
2. the conversion of the data base scheme and perhaps change of the data file received in the previous stage;

3. the loading of a new data base from the serial file.

The idea that a user should specify his restructuring by means of the two non-procedural languages (a "data difinition language" and a "data conversion language") is popular [1].

The method of unloading of a complex network data base structure is known [3]. The main point is to choose several tree structures from the network structure and unload separately each structure into the serial file.

Unloading is realized in the following way: all members are unloaded behind their owner records. The data base key is being recorded in the serial file after each record occurrence. The data base keys of the set member occurrences are being recorded for every set connecting tree structures after unloading of the owner occurrence into the file.

Loading of a new data base is realized in two stages. During the first stage the tree structures are loaded from the file and simultaniously a temporary file of new data base keys indexed by the old data base keys is created. After that the sets which connect the loaded tree structures are created. That is the way the network data base is created.

This approach to unloading and reloading of data bases has a number of disadvantages. The most considerable disadvantage is that one has to carry out a sufficiently deep analysis of the data base scheme in order to find tree structures.

Basic principles of the proposed method for unloading and reloading of data bases

The method proposed in this work is free from the above disadvantage. The diagram explaining the basic principles of the method is given in figure 1. We should point out that each of the stages of unloading and loading of data bases consists of two parts.

Unloading stage

In the first stage all types of data base records are unloaded in text files. The record type unloading program uses information about the types of data base records from

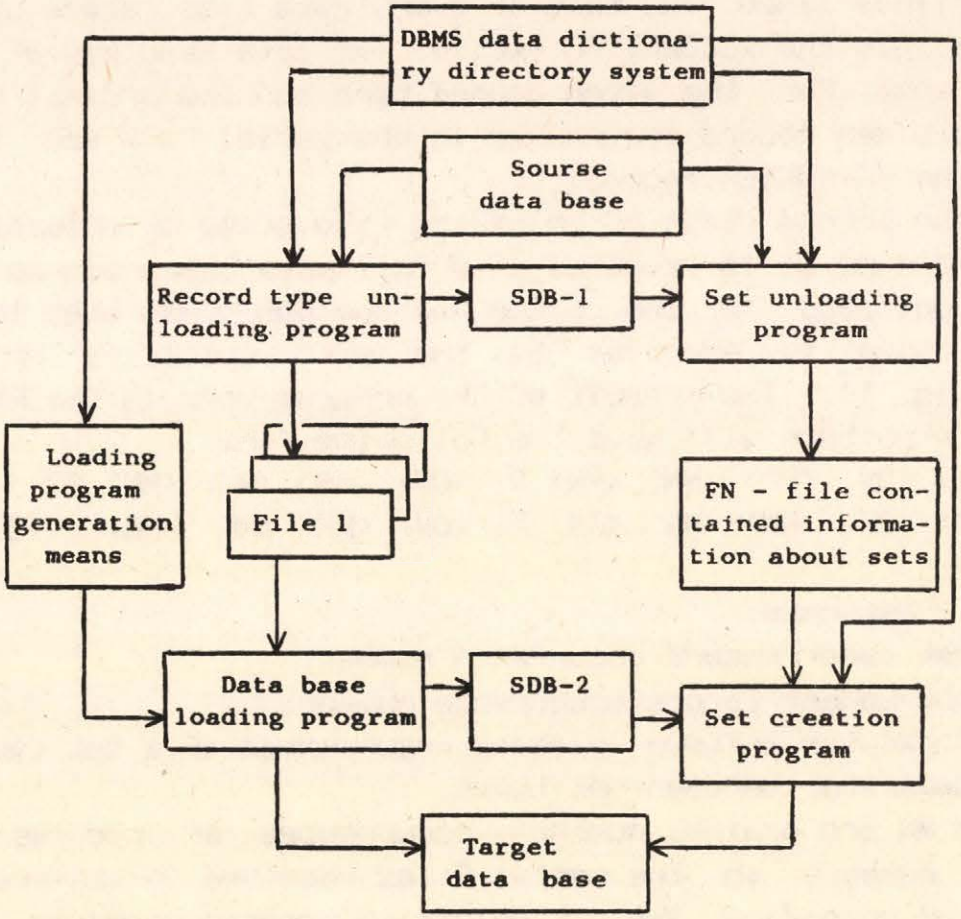


Fig. 1.

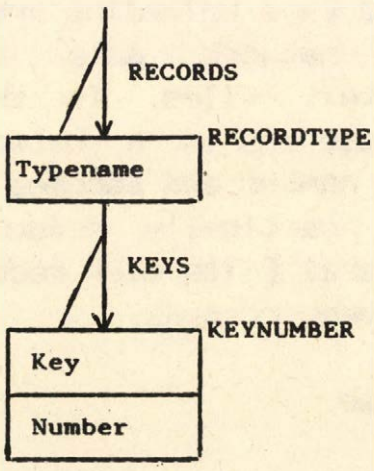


Fig. 2.

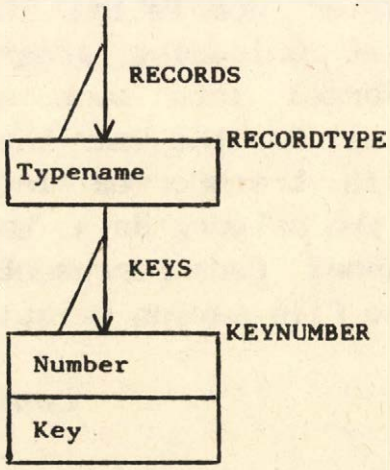


Fig. 3.

the data dictionary directory system. At the same time the special data base N1 (SDB-1) is being loaded. The SDB-1 scheme is shown in figure 2. Set ordering elements are shown by inlined lines. The name of every data base record type is loaded into the RECORDTYPE record. The data base key of every occurrence for the given record type and the ordinal number of the given record occurrence in the serial file are loaded into the KEYNUMBER record.

The second stage of unloading (the stage of unloading of data base sets) is executed after all data base records have been unloaded in the files and the SDB-1 has been loaded. This stage is executed by the set unloading program (see fig. 1). The result of the program work is the FN file which structure will have the following form:

<SNAM1> <ON> <MN> <MN> <MN> R1 <ON> <MN> <MN> <MN> R2
<SNAM2> <ON> <MN> <MN> <MN> R1 <ON> <MN> <MN> <MN>.. <EOF>,

where

SNAME - set name;
ON - set owner record occurrence number;
MN - set member record occurrence number;
R1 - separator between unloaded occurrences of a set type;
R2 - separator between set types.

ON and MN are ordinal numbers, occurrences of records have these numbers in the serial files received in the previous stage of unloading. The set unloading program receives these numbers from SDB-1.

After completing the record type unloading program and the set unloading program the network data base is transformed into some serial text files. In this case invisible for the user the data base key in a network data base is transformed into some number and becomes visible, i. e. the network data base is practically reduced to a relational form representing several files with record types and the file saving relations between records.

Loading stage

The data base loading consists of the following two stages:

1. the loading of all occurrences of all record types

regardless of their set relations;
2. the restoration of the sets in the data base.

Loading begins with the creation of a data base loading program by the DBMS special means.

Before data loading is realized the generated program makes temporary meta-information correction for the data base to be loaded. This correction consists in the following:

1. all "AUTOMATIC" sets are made "MANUAL";
2. for all sets, with owner selection this selection is abolished;
3. for loop sets the set owner selection is made by the data base key of the owner.

This correction acts only during the work of loading program. After that the data base loading of the information received in the unloading stage is executed. Simultaneously the loading of the special data base N2 (SDB-2) which scheme is shown in figure 3 is being executed. The only difference between SDB-2 and SDB-1 is the KEYS set ordered by the NUMBER element. The set creation program is intended for relating record occurrences into sets (see fig. 1). This program uses the FN file and the DBMS data dictionary directory system. For every set the set creation program finds the data base keys of the owner and member in SDB-2 by numbers from the FN file and connects the records with these data base keys. After the set creation program is completed the new data base will be received.

Application of the method for restructuring of network data bases

One may choose the following forms of data base scheme restrusturing:

1. the change of insertion mode in a set;
2. the change of disconnection mode from a set;
3. the change of set selection mode;
4. the creation of record type;
5. the record type deletion;
6. the creation of an element of existed record;
7. the record element deletion;
8. the creation of a set (owner and member record

types exists in the data base);

9. the set type deletion with saving owner and member record types;
10. the change of set ordering.

It will be noted that the first four kinds of data base scheme restructuring need no physical data base reorganization. In order to execute any of the first four kinds of restructuring it is enough to retranslate a new description of the data base scheme and to create new subschemes.

In most DBMSs it is impossible to execute other kinds of scheme change without physical data base reorganization. But the execution of it offers no difficulties, if the method of unloading and reloading suggested in the work is used. It is enough to unload the part of the data base which is being restructured, to change data base scheme, to create a new subscheme or subschemes and to load the data base.

In all the cases mentioned above the data base scheme and subscheme changes are executed by standard DBMS means. All other restructuring actions are automatically executed by the data base loading program and set creation program.

Merits of the proposed method

First of all it is necessary to note a sufficiently high degree of automation of the proposed method. The data base unloading and loading programs are adjusted to the user subscheme and all necessary actions are automatically executed. All the user has to do is practically to make the necessary changes in the data base scheme (in case they are needed).

In comparison with the traditional approach to reloading and restructuring the proposed method does not require any complex analysis of the data base scheme and does not require saving of the old data base keys in the process of creating a new data base.

Data base key conversion to an abstract ordinal number makes the data base key independent of the type of the computer. Together with the unloading of a data base to a series of serial files it ensures the simplicity of the data

base transfer to the other kinds of computers.

The condition of the unloaded data base will be more compact than unfolded condition since the indexes by means of which the sets are created are not unloaded into the file. Thus it will be convenient to use the unloaded data base presentation for the creation of data base archives.

It should be noted in the conclusion that the method is universal and can be easily applied to different network DBMSs.

References

1. M.E. Deppe, J.P. Fry and D.E. Swartwout, Operational software for restructuring network data bases, Proc. AFIPS NCC 46 (1977) 499.
2. V.I. Filippov, Data base management system "Compass", in: Problems in design and application of data base systems, (Univ. of Turku, Turku, Finland, 1980) 61-72.
3. M.A. Steel, Automating the restructuring of network data bases, The Australian Computer Journal, vol. 13, No. 4, (1981) 109-113.

IMYCS'88, Smolenice Castle, November 14-18, 1988.

ON SOME DETERMINISTIC GRAMMARS WITH REGULATION

HENNING BORDIHN

Technological University
Magdeburg
Department of Mathematics
PSF 124
DDR - 3010 Magdeburg
German Democratic Republic

ABSTRACT: The hierarchy of language families generated by deterministic types of pure context-free, pure matrix, pure programmed and pure random context grammars with and without erasing rules is determined. Further we locate these classes in the hierarchy of language families generated by the corresponding nondeterministic versions.

0. INTRODUCTION

The most investigated language class is the context-free one. However, a lot of aspects which are of interest (e.g. in the theory of programming languages) one cannot describe by context-free grammars only. Therefore a series of grammars with regulated rewriting and context-free core rules has been introduced.

In /1/ J. Dassow explored the hierarchy of some pure language families of regulated rewriting in order to determine the differences between the mechanisms which often disappear if we consider sets of words over a terminal alphabet, since all

the intermediate steps of the derivation are disregarded. Further, these pure languages are of interest for themselves by the following two facts:

- the intermediate steps are important for the syntax analysis,
- pure grammars form a sequential counterpart to the L systems (with regulation).

Hence it is natural to ask for the hierarchy of language classes generated by deterministic versions of those grammars with regulated rewriting. This is the main goal of this paper where especially the following three types of regulated devices are considered: matrix grammars, programmed grammars, and random context grammars.

We assume that the readers are familiar with the basic notions of formal language theory and have some information on regulated rewriting (e.g. see /2/, /3/).

1. DEFINITIONS

We give the formal definitions of the pure versions of the above mentioned grammars. In the following definitions, let V be an alphabet, and let S be a finite subset of V^+.

A pure context-free grammar is a triple $G = (V, P, S)$ where P is a finite subset of $V \times V^*$ (as usually, its elements are written as $a \to w$ instead of (a,w)). We say that $x \in V^+$ directly derives $y \in V^*$ in G (written as $x \Rightarrow y$) iff

$x = x_1 a x_2$ for some $a \in V$, $x_1, x_2 \in V^*$, and $y = x_1 w x_2$, and there is a rule $a \to w \in P$.

$\overset{*}{\Rightarrow}$ denotes the reflexive and transitive closure of $\Rightarrow$. The language generated by G is defined as

$$L(G) = \{ y : z \overset{*}{\Rightarrow} y \text{ for some } z \in S \}.$$

A pure matrix grammar is a quadruple $G = (V, M, S, F)$ where M is a finite set of finite sequences of productions,

$$M = \{ m_1, m_2, \ldots, m_t \},$$

$$m_i = (a_{i1} \to w_{i1}, a_{i2} \to w_{i2}, \ldots, a_{ir(i)} \to w_{ir(i)}),$$

$a_{ij} \in V$, $w_{ij} \in V^*$, $1 \leq i \leq t$, $1 \leq j \leq r(i)$, and F is a subset of occurrences of rules in M.

Then, for $1 \leq i \leq t$, we say that $x \underset{m_i}{\Longrightarrow} y$ iff

$$x = y_0 \Rightarrow y_1 \Rightarrow y_2 \Rightarrow \dots \Rightarrow y_{r(i)} = y$$

where

$$y_{j-1} = z_{j1} a_{ij} z_{j2}, \quad y_j = z_{j1} w_{ij} z_{j2}$$

for some $z_{j1}, z_{j2} \in V^*$ or

a_{ij} does not occur in y_{j-1}, $a_{ij} \rightarrow w_{ij} \in F$, and $y_j = y_{j-1}$.

The language $L(G)$ generated by G is defined as the set of all words y which are obtained by iterated applications of matrices (elements of M) to words of S and all intermediate words (y_i in the above notation) of these applications of matrices in M. If we need occurrences of rules in F for the rewriting process we use the so-called appearance checking mode. Otherwise, iff $F = \emptyset$ we say that G is defined without appearance checking.

By $P(M)$ we denote the set of all occurrences of matrices in M.

A <u>pure programmed grammar</u> is a triple $G = (V, P, S)$ where P is a finite set of rules of the form

$$(b: a \rightarrow w, E(b), F(b))$$

where b is a label of the production, $a \in V$, $w \in V^*$, and $E(b)$ and $F(b)$ are subsets of the set of labels.

The language $L(G)$ consists of all words y such that there is a derivation

$$z = y_0 \underset{b_1}{\Longrightarrow} y_1 \underset{b_2}{\Longrightarrow} y_2 \underset{b_3}{\Longrightarrow} \dots \underset{b_{n-1}}{\Longrightarrow} y_{n-1} \underset{b_n}{\Longrightarrow} y_n = y$$

where $z \in S$, $(b_i: a_i \rightarrow w_i, E_i, F_i)$ are rules in P, $1 \leq i \leq n$, and for $1 \leq i \leq n$,

$$y_{i-1} = z_{i1} a_i z_{i2}, \quad y_i = z_{i1} w_i z_{i2} \text{ for some } z_{i1}, z_{i2} \in V^*,$$

and $b_{i+1} \in E_i$ (if $i < n$)

or

a_i does not occur in y_{i-1}, $y_i = y_{i-1}$, and $b_{i+1} \in F_i$.

Whenever $F(b) = \emptyset$ for all productions in I, we have a grammar without appearance checking, again.

A pure random context grammar is a triple $G = (V, P, S)$ where P is a finite set of productions of the form

$$(a \to w, Q, R),\ a \in V,\ w \in V^*,\ Q \subseteq V,\ R \subseteq V\ .\ .$$

We say that $x \Rightarrow y$ iff

$$x = z_1 a z_2,\ y = z_1 w z_2,\ (a \to w, Q, R) \in P$$

$z_1, z_2 \in V^*$, $z_1 z_2$ contains all letters of Q, and $z_1 z_2$ contains no letter of R.

The language $L(G)$ is defined as

$$L(G) = \{ y : z \overset{*}{\Rightarrow} y \text{ for some } z \in S \},$$

again.

Analogously, we say that a random context grammar is without appearance checking iff $R = \emptyset$ for all productions in P.

These definitions are the most general ones, i.e. rules of the form $a \to \lambda$ are allowed. A grammar is refered to be λ-free iff λ is forbidden on the right hand side of any core rule.

We set

$$(pCF) = \{ L(G) : G \text{ is a pure context-free grammar} \}$$

$$(pCF \setminus \lambda) = \{ L(G) : G \text{ is a pure context-free } \lambda\text{-free grammar} \}.$$

By (pM, CF, ac), (pP, CF, ac), (pRC, CF, ac) we denote the families of languages obtained by pure matrix, pure programmed, and pure random context grammars, respectively. If we omit ac in this notation, we consider the corresponding families generated by grammars without appearance checking. If we have λ-free grammars, we write $CF \setminus \lambda$ instead of CF, again.

A grammar is called deterministic iff it satisfies the following condition: If $a \to w_1$ and $a \to w_2$ are core rules occurring in P or $P(M)$, then $w_1 = w_2$.

By (pDCF), (pDCF$\setminus\lambda$), (pDX, Y, β) for $X \in \{M, P, RC\}$, $Y \in \{CF, CF \setminus \lambda\}$, β = ac or empty we denote the corresponding families of languages generated by the deterministic versions of these grammars.

2. RESULTS

The main result of the paper will be presented in a figure. There is used the following notation: Whenever there is an arrow between two language families, the language family at the head of the arrow properly contains the language family at the base of the arrow. If there is no explicit path from one language family to another, these language families are incomparable.

Theorem: The hierarchy shown in Figure 1 holds.

Concerning the language family (pDCF) which is not contained in Figure 1 we have the following facts:

(i) $(pDCF \setminus \lambda) \subseteq (pDCF) \subset (pCF \setminus \lambda) = (pCF)$.

(ii) For $X \in \{M, P, RC\}$, we have
$(pDCF) \subset (pDX, CF)$.

We note that the following problems are open:

1. Is the inclusion $(pDCF \setminus \lambda) \subseteq (pDCF)$ proper?
2. It is not known whether, for $X \in \{M, P, RC\}$, β = ac or empty, $(pDCF) \setminus (pDX, CF \setminus \lambda, \beta) \neq \emptyset$ or
$(pDCF) \subseteq (pDX, CF \setminus \lambda, \beta)$ hold.

The proof of the Theorem follows by a combination of lemmas of the following and similar form.

Lemma: For $X \in \{M, P, RC\}$,
$(pCF \setminus \lambda) \setminus (pDX, CF, ac) \neq \emptyset$.

Proof of the Lemma: We consider the language

$$L_o = \{ xcx^R : x \in \{a,b\}^* \}$$

where x^R is the mirror image of x. Obviously, the pure context-free grammar

$$G_o = (\{a,b,c\}, \{c \to aca,\ c \to bcb\}, \{c\})$$

generates L_o. Therefore $L_o \in (pCF \setminus \lambda)$.

$L_o \notin (pDM, CF, ac)$. Let us assume that $L_o = L(G)$ for some pure deterministic matrix grammar $G = (V, P, S, F)$ with context-free core rules. We consider the word $a^n c a^n$ for a sufficiently large integer n such that $a^n c a^n \notin S$. Then

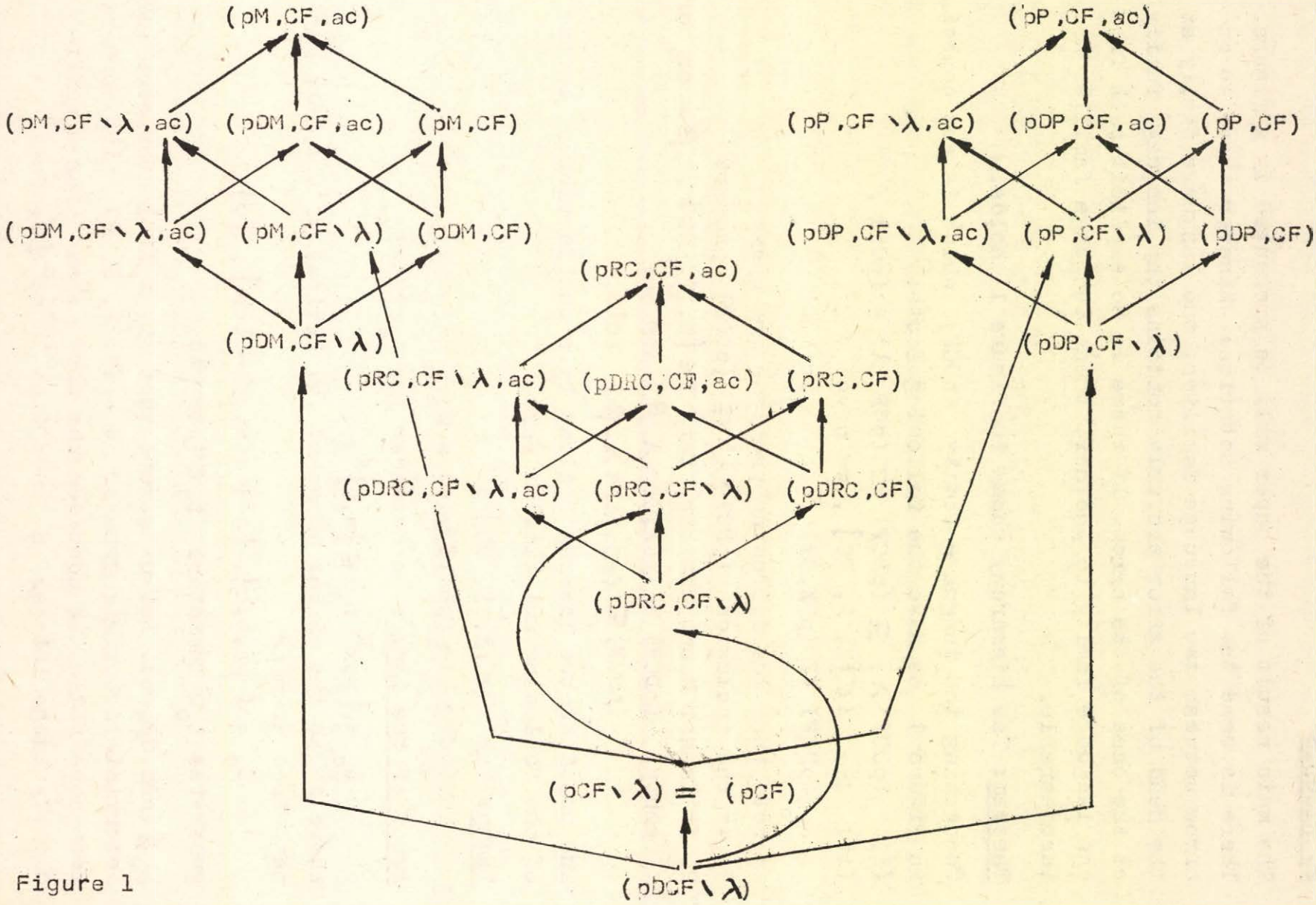

(pM,CF,ac)
(pM,CF\λ,ac)
(pDM,CF,ac)
(pM,CF)
(pDM,CF\λ,ac)
(pM,CF\λ)
(pDM,CF)
(pDM,CF\λ)
(pP,CF,ac)
(pP,CF\λ,ac)
(pDP,CF,ac)
(pP,CF)
(pDP,CF\λ,ac)
(pP,CF\λ)
(pDP,CF)
(pDP,CF\λ)
(pRC,CF,ac)
(pRC,CF\λ,ac)
(pDRC,CF,ac)
(pRC,CF)
(pDRC,CF\λ,ac)
(pRC,CF\λ)
(pDRC,CF)
(pDRC,CF\λ)
(pCF\λ) = (pCF)
(pDCF\λ)

Figure 1

there has to exist a word $z \in L_o$ which directly produces $a^n c a^n \notin S$. Since we can rewrite only one symbol in each derivation step, z has to be of the form $a^r c a^r$, $r < n$ ($r > n$ is impossible since only one symbol can be rewritten, and $r = n$ is not of interest since we can assume that $z \neq a^n c a^n$). Then we applied a rule of the form $c \rightarrow a^s c a^s$, $s \geq 1$, to z.

Now let us consider a word $b^m c b^m$ where m is chosen such that $b^m c b^m$ is not an axiom, again. By the same arguments as above we can verify that there has to be a rule of the form $c \rightarrow b^t c b^t$, $t \geq 1$, in P(M). This contradicts the determinism of G.

$L_o \notin$ (pDP, CF, ac) and $L_o \notin$ (pDRC, CF, ac) can be proved analogously.

REFERENCES

/1/ J. Dassow, Pure languages of regulated rewriting and their codings (To appear in Acta Cybernetica).

/2/ O. Mayer, Some restrictive devices for context free grammars, Inform. Control 20 (1972) 69 - 92 .

/3/ A. Salomaa, Formal Languages (Academic Press, 1973).

IMYCS'88, Smolenice Castle, November 14-18, 1988.

THE SYNCHRONIZATION AND MESSAGE EXCHANGE MECHANISM IN THE REAL TIME DISTRIBUTED OPERATING SYSTEM PARUS

D. I. BUYANOVSKI AND A. A. MENN
Institute of Control Science, Moscow, USSR

ABSTRACT: The paper deals with the description of synchronization mechanism and that of message exchange between processes in the distributed real-time system PARUS. Processes communicate through local ports whose links are described at an additionally introduced stage of system development. The PARUS allows processes to wait for situations - complex logical combinations of events - which can essentially increase the efficiency of an operating system.

1. INTRODUCTION

Communication and synchronization mechanisms significantly affect the efficiency of a real-time distributed operating system, DOS, and feasibility of autonomous designing its separate components. Therefore, in designing the PARUS DOS [1] great attention has been paid to

communication and synchronization means.

It is known that a large program is most conveniently developed fractionally. To achieve this, it should be divided into components and each of them is to be programmed and debugged, then the program is to be constructed as a configuration of these components [2]. PARUS offers means for developing separate components (Pascal-like component programming language) and those for their correct joining (configuration language). Components (processes) are developed separately and independently from the environment in which they are to be used; in other words, the components do not need data for adapting them to specific stations or partners. This information is contained in the configuration program.

The processes communicate through local ports. This makes it possible to receive and send messages without specifying sender and receiver addresses. This is the main advantage of communicating through the ports. Consequently, for development of a process interface with the others processes the semantics and formats of messages in local ports are sufficient. The local ports of the processes are linked in the configuration program. Linking the local ports means defining input ports where messages written in the output ports are to be delivered.

The DOS efficiency is of great importance in many real-time systems. The proposed mechanism of situations and that of packet transmission make PARUS highly efficient.

2. PORTS

In a number of existing systems the processes or tasks communicate synchronously. With a message having been sent or an entry called, the sending process waits till the sent message is received (and even processed in the Ada system) by the communication partner. The basic advantage of synchronous communication is belived to be the ease of

understanding and checking the program functioning [3]. Synchronous communication constrains, however, the potential parallelism of process execution and does not support broadcasts although modern communication channels make broadcasting possible. Programming real-time applications using synchronous communication requires additional intermediary processes to realize asynchronous communication [3]. As a result, the total overhead of the processor time for synchronization, communication, and process context switching becomes inadmissible for real-time systems.

PARUS supports the asynchronous interaction scheme of processes. It is due to the fact that this scheme is free of the above-mentioned disadvantages, and, if necessary, the synchronous communication can be efficiently implemented by an asynchronous one. In asynchronous communication the execution of the process which has sent a message is continued whether or not this message is expected by the receiver. The processes exchange messages through ports. The ports are classified into input and output ones. A process sends messages into the environment through output ports and receives messages from the environment through input ports.

The messages transmitted to a process are buffered in its input ports. The process may receive these messages from the ports using conditional or unconditional requests. If there are already buffered messages in one of the ports, then both conditional and unconditional requests cause one of them to be chosen and transmitted to the process. If there are no buffered messages, then unconditional requests result in blocking the process until a message arrives in the port, while conditional requests allow its continuation. Thus, unconditional requests may be used when the arrival of a message has to be waited for, and conditional requests when the execution may be continued even if no message has arrived.

Input ports incorporate packets for buffering arriving

messages. The number of packets is limited for every port and is defined by the port description in the configuration program. Packets with buffered messages are referred to as filled, and the others as empty. The filled packets are transmitted from the port to the process either in FIFO (as they arrive) or LIFO (inverse) order. If a message has arrived to the port and there are no empty packets available, the message is rejected in the FIFO port and buffered by erasing the earliest message in LIFO port.

When receiving a message in a process data area the packet (buffer) address rather than its content may be copied. As a result, the process receives the packet itself. Once the data in the packet has been used, the process returns it into the port as an empty one (so as not to deprive the port of buffers). In the case of long messages the elimination of their rewriting from the buffer into the process data area improves the system efficiency.

The output port has a single packet. To send a message into the port, process should have a reference to that packet, write a message in it and send it to the output port. If the communicating input and output ports are in the same station, then, whatever the message length, the communication is an exchange of the addresses of the filled packet from output port and the empty packet from the input port. In communicating the ports simply exchange packets. Note that direct packet exchange is only feasible with one communicant while for communication with the others the message has to be rewritten.

The links between ports are established in the configuration program statically. Several output ports may be linked to several input ports. Therefore it is necessary to support the so-called library processes which can receive initial data from several partners and return the results of processing this data to the sender.

For library processes the readdressed port mechanism

has been introduced in PARUS. The port is referred to as readdressed if it is not statically linked with other ports but is in the readdressing list of some port. In communicating ports with nonempty readdressing lists the links between appropriate ports of these lists are established automatically. The readdressing list of output port may include only an input port and that of the input port, only the output port of the same process.

In the existing systems where components communicate through ports the library process design is facilitated by including additional request-reply transactions [3]. In PARUS the readdressed port mechanism makes it possible for the components to send and receive replies as usual messages because the readdressing lists are defined in the configuration program rather than in components. Consequently, the environment for the process is completely described by semantics and formats of received and transmitted messages and there is no necessity to distinguish between messages and replies to messages.

3. SITUATIONS

The processes, as a rule, are cyclic programs. In a general case for execution of every cycle it is necessary to have input data generated by several other processes. Communicating only through ports, the processes may need messages from several input ports. In this case the process is blocked until the messages needed arrive.

In a distributed system various communication objects are employed for the processes to communicate. In every case, however, the process may wait for at one time one data only from one communication object or an alternative group of these objects. Early in the cycle, n process activations and blockings may be required for awaiting data from n communication objects because it is necessary to declare n times that data are expected, if these data have not arrived

the process is blocked and when they do it is activated. Blocking and activating consume much time and their execution on every occasion is undesirable for a real-time DOS. The situation mechanism in PARUS eliminates redundant activations. The process may receive n messages and be blocked just once. The situation mechanism is useful when the process needs one or several messages from a group of ports in order to execute the cycle.

A situation is determined as a logical combination of local events in the process. A local event for the process is either reception of a message from a port or the elapsing of a time interval. A process without intermediate activations may await an arbitrary situation which is represented in the conjunctive normal form as

A_0 & A_1 & .. & A_k

where A_i is $(e_{i0}$ V e_{i1} V .. V $e_{in})$ and e_{ij} is an event.

The range of situations is rather wide. Thus expressions such as

(a & b) V (c & d)

may be transformed into conjunctive form,

(a V c) & (b V c) & (a V d) & (b V d).

In waiting for a situation every event e_{ij} is assigned a binary code with the i-th bit set and the others bits cleared (event e_{ij} is encoded as 2^i). If an event is specified several times in a situation, then it is encoded independently every time and the resultant code of the event is obtained by the logical "OR" of these codes. What is important is that in different situations the same event may be encoded in different ways. The situation is assigned a code which is the logical "OR" of codes of events in it. Occurrence of an event is marked by clearing the bits in the situation code to which set bits correspond in the event code (this is done by one computer instruction). The situation is satisfied when its code is cleared.

Thus, situation formulas can be translated into situation codes before the DOS starts operating rather than

interpreted at run-time. In this way the implementation of the situation mechanism is significantly facilitated and its application field is expanded.

In numerous systems [2, 3] the statements of alternative input from several communication objects make it possible to access options which are currently acceptable. Thus the SELECT statment of Ada does this by guarded commands. The situations mechanism also makes it possible to dynamically eliminate events from the situation depending on the conditions at run-time.

4. CONCLUSION

The paper contains consideration of the situation mechanism and that of the readdressed ports which are realized in the PARUS operating distributed system. The former coordinates efficiently and flexibly the execution of processes in a distributed system and latter stringently controls library processes in communication through ports.

REFERENCES

1. A. I. Kaz'min, A. A. Menn, D. I. Buyanovski, et al., The Distributed Operating System PARUS (Moscow, Institute of Control Sciences, 1987).

2. M. Sloman, J. Kramer, J. Magee, K. Twiddle, Flexible Communication Structure for Distributed Embedded Systems, IEE Proceedings, Pt. E 133(1986) 201-211.

3. S. J. Young, Real Time Languages: Design and Development (New York, Elis Horwood, 1982).

IMYCS'88, Smolenice Castle, November 14-18, 1988.

AN EXTENSION OF THE GRID FILE TO INCREASE THE EFFICIENCY OF DATA RETRIEVAL WHEN RANGE OR PARTIAL-MATCH QUERIES ARE PERFORMED

DANIELE CORTOLEZZIS

Algotech s.r.l., via Biella,10, I-00182 Rome, Italy

ABSTRACT: In this paper, an extension of the grid-file structure is introduced which makes it possible to substantially decrease the number of disk accesses performed during the resolution of range or partial-match queries. An analysis of such decrease is presented for both the general and a particular case.

1. INTRODUCTION

Dealing with a great amount of information specified by multiple access key records (multidimensional data) and stored in buckets on mass memory (such as disk) has put in evidence the weakness of traditional data structures applied to interactive information systems which frequently require to perform insertions, deletions and to answer to exact-match queries, range queries and partial-match queries.

In this context, the efficiency of a data structure in storing, searching and updating information on disk is measured in terms of the number of disk accesses necessary to perform a single operation, where each access involves a certain amount of information, stored in the same bucket.

Moreover, to answer range queries, the structures have to respect "as much as possible" the adjacency relation (contiguity).

In other words, records associated with keys which are contiguous in the logical key space, must be stored in the same or in contiguous buckets.

In such a way a range query involving **n** records could perform, in the best case, **n/b** accesses to retrieve information (where **b** is the bucket size).
This depends on the query structure and on the key distribution in the logical key space.

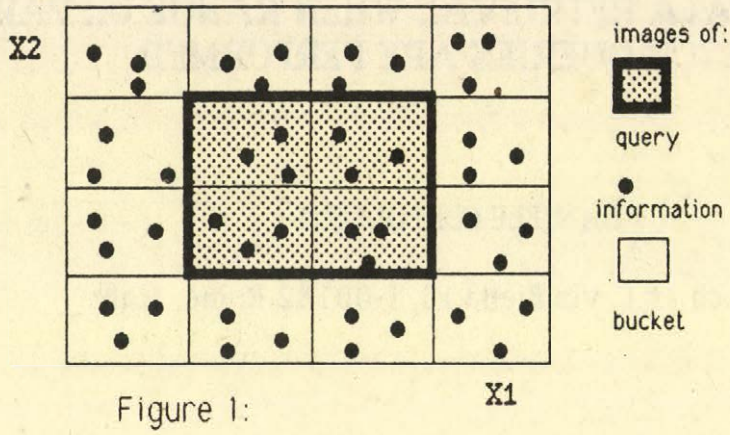


Figure 1:

In figure 1, an example of a query with this good performance is shown.
Data are identified by two keys (X_1,X_2) and if two data are adjacent in the logical key space, they are stored in the same or in two adjacent buckets, in this case the rectangle query selecting **n** records requires **n/b** accesses.
The performance of the data structures for range or partial-match queries on disk depends on the capability to realize and preserve contiguity of adjacent data in the logical key space.
This restriction strongly limits the use of tree structures[6][7]: for example, the multidimensional-B-trees [4], introduced as extensions of B-trees, are inefficient for what concerns the execution of queries which involve sets of buckets containing keys adjacent in the logical key space.
In this case, since buckets are stored on tree nodes, in order to retrieve all required information it is necessary to visit many times the tree structure.
Moreover, neighbor keys may not be stored in neighbor subtrees, because the multidimensional-B-tree cannot represent a total order on the logical key space.
The same considerations can be derived for other tree-structures such as k-d trees [8], R-trees [10], quad-trees [11].
As far as trie structures are concerned, it is possible to observe that they fail when partial-match queries are considered [6][7], while inverted files are ineffective when we perform range queries involving wide interval on secondary keys [6][7].
The most efficient structure for this kind of problems is the grid-file [1][3][5][6][7][9]: it generalizes the standard single key retrieval hash techniques to multidimensional data, assuring the contiguity of the stored information and answering to exact match queries with a single access to mass memory.

But the grid-file can turn out to be inefficient for particular range queries, since it imposes a fixed partition on the information: in figure 2 we can see an example in which **n** is the number of accesses to retrieve **n** records.

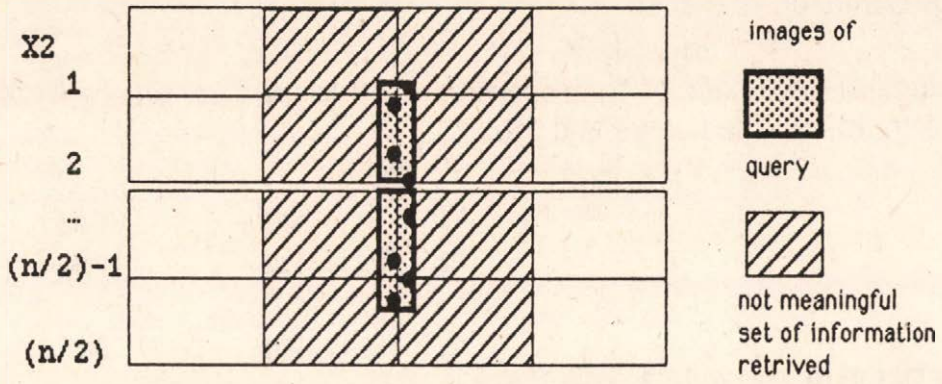


Figure 2

This poor behaviour depends on the fact that the considered query is represented by a rectangle in which the perimeter and the surface involve the same number of buckets and, thus, forces the system to transfer in central memory a large fraction of non meaningful data.

This situation induces a loss in performance proportional to the increase of the rate between non meaningful and meaningful information.

Figure 3 presents this situation: here the non meaningful set has less influence on the performance because the number of accesses tends to **n/b** anyway.

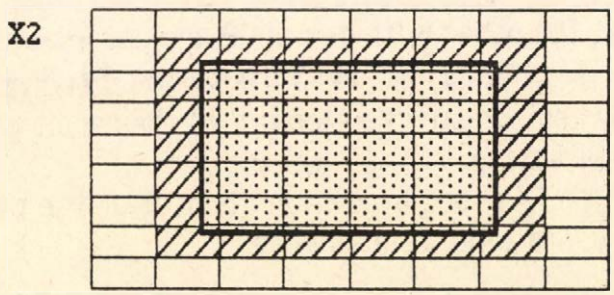


Figure 3:

In this paper we present a data structure that cuts down the amount of meaningless information retrieved by a query, in order to limit the number of disk accesses.

Furthermore, we will compare the behaviour of the new structure with the original grid-file both in the worst and in the average case.

2. GRID-FILE STRUCTURE

The grid-file is conceptually based on a partition of the logical key space into subspaces which are associated to buckets, stored in mass memory, where information is placed.

Let $S = X_1 \times X_2 \times \ldots \times X_N$

be the logical key space of the records stored in the file, where $X_1, X_2, \ldots, X_N$ are the domains of the N keys and let

$$P = P_1 \times P_2 \times \ldots \times P_N$$

where $P_1 = \{p_{11}, p_{12}, \ldots, p_{1k_1}\}$

$P_2 = \{p_{21}, p_{22}, \ldots, p_{2k_2}\}$

$\ldots$

$P_N = \{p_{N1}, p_{N2}, \ldots, p_{Nk_n}\}$

be a partition of the logical key space.

In such partition P_J represents the set of values that identify the borders of the intervals in which the j-th dimension is partitioned.

Moreover, let us denote as M the set of references to disk buckets.

Now, the grid-file manager system carries out a function which maps each element of the logical key space to an element of M, or alternatively which joins the single information image on S to the reference to the bucket that stores it.

$F : S \dashrightarrow M \quad ; \quad F : (x_1, x_2, \ldots, x_N) \mapsto \mu$

where μ: reference to the bucket containing the record identified by the key values $(x_1, x_2, \ldots, x_N)$.

This function is defined as the composition of two functions, the first with domain S and codomain P, the other with domain P and codomain M.

Moreover, function F_1 is compounded by N functions $F_{11}, F_{12}, \ldots, F_{1N}$ that determine, component by component, the element of the partition to which the information referenced by $(x_1, x_2, \ldots, x_N)$ belongs.

On the other side, function F_2 associates each element of the partition to the reference to the bucket containing its elements.

$F = F_1 \circ F_2 \quad ; \quad F_1 : S \dashrightarrow P \quad ; \quad F_2 : P \dashrightarrow M$

$F_1 (x_1, x_2, \ldots, x_N) = (p_1, p_2, \ldots, p_N) \; ; \; F_2 (p_1, p_2, \ldots, p_N) = \mu \; ;$

$F_1 (x_1, x_2, \ldots, x_N) = (F_{11}(x_1), F_{12}(x_2), \ldots, F_{1N}(x_N)) = (p_1, p_2, \ldots, p_N)$

where p_J: compact representation of p_{Ji}.

The function F, to which we refer as "find-function", is physically represented by N dynamic arrays (for $F_{11}, F_{12}, \ldots, F_{1N}$)

$$Y_1[1..K_1], \; Y_2[1..K_2], \; \ldots, \; Y_N[1..K_N]$$

in which the generic cell $Y_H[J]$ contains the upper limit of the J-th partition element according to H-th dimension of the space (p_{HJ}), and by one dynamic N-dimensional matrix for F_2

$$G\,[1..K_1, 1..K_2, \ldots, 1..K_N]$$

in which every cell contains a reference to mass memory.

Finally, we can observe that the find-function first determines all p_J by a search on arrays Y_j, then, by the look up table G, carries out the reference to the bucket containing the required information (of course if the data is not present then the system carries out the bucket in which the information would be stored).

It is worth noting that there is no bijective correspondence between P's elements and M's elements: the grid-file imposes that all records belonging to the same partition's element have to be stored in the same bucket, but records stored in the same bucket may belong to different partition's element.

This solution is imposed to avoid bucket proliferation in mass memory with a small occupation rate.

Let us now briefly consider how the typical operations are performed on a grid-file.

2.1 INSERTION

If a new record has to be inserted, the find-function determines what is the bucket interested by the insertion.

If the insertion does not cause a bucket overflow, the operation ends normally; on the contrary, if it causes a bucket overflow, we must determine a splitting-value in order to share in two buckets the information contained in the full one.

The choice of the splitting-value differs in dependance of which kind of file management policy is adopted [2][7][9] and depends on whether only one or more references of G point to the full bucket[7][2].

In the first case some of such references will be used to point the new bucket, otherwise, the grid-file management system will provide to spread the arrays properly.

2.2 DELETION

This operation evolves in a dual way with respect to insertion: if two border buckets have less information than the capacity of a single bucket, then the system merges the data contained in the buckets and, eventually, contracts the directory.

2.3 FIND

The method used to find a single information (answer to a exact-match query) is fully explained in the theoretical presentation of grid-file above.

Therefore we will consider only the procedures the system uses to answer to range and partial-match queries.

If we have to retrieve information in order to answer a partial-match query, whose image mapped on the logical key space is the subspace orthogonal to the dimensions relative to the keys specificated in the query, then we must gather all the buckets whose references belong to the cells of G whose indices are $F1U_1(x_{U1}), \ldots, F1U_C(x_{UC})$.

On the other hand, if a range query is to be performed, a N-dimensional hyper-rectangle is specified and all buckets are to be gathered whose image on the logical key space intersected the query's image.

From the definition of the last two kinds of queries it is possible to note that the partial-match query is indeed a particular range-query in which some dimensions are bounded and the others have the same range of the space. In the continuation we will treat them in the same manner.

3. A MODIFIED GRID-FILE TO INCREASE THE EFFICIENCY OF INFORMATION RETRIEVAL.

Since, in order to answer a range-query, the system is often forced to transfer in main memory a set of not meaningful data, our aim is to minimize such redundant set.

Figure 2 presents such a kind of situation, that becomes critical if we work on an interactive application in which queries involve only few buckets that are characterized to have the area of the same size of the perimeter, measured in number of buckets involved.

In order to tackle this problem theoretically we model the shift of a floating grid to frame the query's rectangle into a minimal number of N-dimensional intervals.

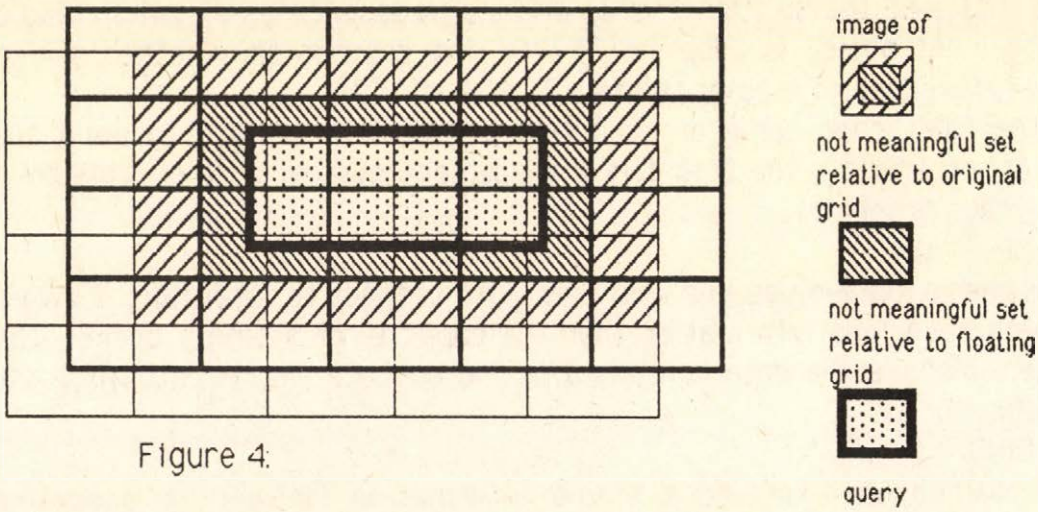


Figure 4.

Physically this operation would require the duplication of data stored in the file and the creation of a new logical key space partition.

It involves that the system determines new different splitting-values, so that no new bucket contains the same information of the buckets of the first partition.

This results in a certain number of file replies (maintained at insertion/deletion time) which approximate the ideal solution considered above.

The implementation of such approach depends on the adopted bucket management policy.

If Nievergelt's approach is considered [7] the determination of splitting-values is performed by choosing the median value between the splitting-values previously determined, independently from the key distribution on the space.

In this case we determine the new grid splitting-values shifting the original grid splitting-values of a fixed quantity δ.

Now, in order to evaluate the number of accesses the method allows to save, we give the following lemma, whose proof is given in [2]:

Lemma 1

Given a function $QM = \sum_{i=1}^{K} q_i \sum_{j=1}^{i} q_j$, such that $0 \le q \le 1$ and $\sum_{h=1}^{K} q_h = 1$, the value of QM is maximum iff all q_h elements are equal. In other words

$$QM \text{ is maximum } \langle === \rangle \ \forall\, I,J \ \ q_I = q_J = 1/K \ .$$

Theorem 1

If a one-dimensional space with uniform key distribution is considered, the average save it is possible to obtain through the reply and distribution of information in K grids is

$$QM = (K-1)/2K$$

Sketch of proof:

In the first step let us define in mathematical terms the average save, clearly, the query's image onto a one-dimensional space is an interval.

The number of buckets that it is necessary to draw in order to answer to a query, in relation with the R-th grid, is given by the following expression

$$n_R = L + Z_R(p_1,p_2) = [(l/s)+f_{min}] + 1 + Z_R(p_1,p_2)$$

where l: width of the query (defined by the measure unit of dimension);

s: width of the pages (defined in the same unit; the bucket's images are isomorphic because a uniform key distribution is considered [2].

$f_{min} \quad \min_{1 \le R \le K} (f_R)$, where f_R: distance between p_1 and the R-th grid splitting-value at the left to p_1;

Z_R: that we will call invasion-function, is defined as follows: $Z_R=0$ if the interval is contained in $[(l/s)+f_{min}]+1$ contiguous elements of R-th partition, while $Z_R=1$ if the query image invades the (L+1)-th element too.

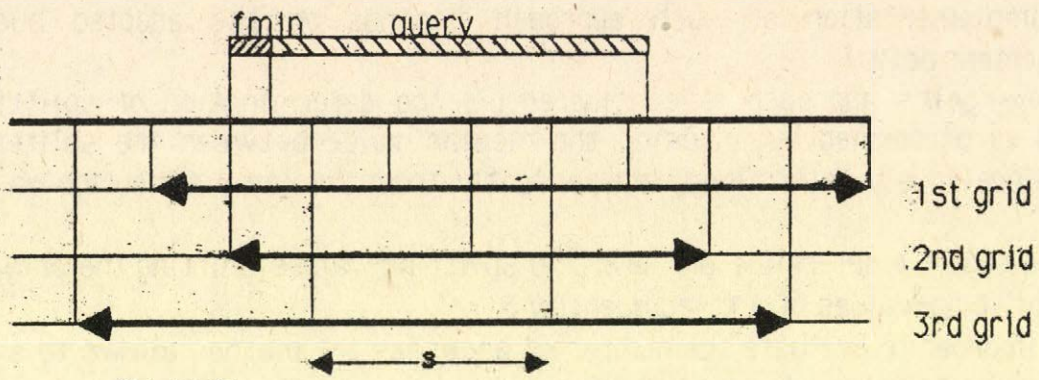


Figure 5

By observing figure 5 it is possible to note that each grid identifies K subsets for each original partition element, whose border is marked by the generic partition splitting-value inside the original partition element.

We will call C_1 the set of intervals included between the splitting-values of original grid (r_1) and the second grid ones (r_2), C_R the one in which the elements are the intervals included between r_R and r_{R+1} ($r_R < p_1 \leq r_{R+1}$).

Also we can express the invasion-function in relation with p_1 and p_2.

$$Z_R(p_1,p_2) = \begin{cases} 0 & \text{if} \quad r_R < p_1 \leq r_{R+1} \;\&\; r_1 < p_2 \leq r_1 + s \\ 0 & \text{if} \quad r_J < p_1 \leq r_{J+1} \;\&\; r_1 < p_2 \leq r_R \;\&\; J < R \\ 1 & \text{if} \quad r_J < p_1 \leq r_{J+1} \;\&\; r_R < p_2 \leq r_J \;\&\; J < R \\ 0 & \text{if} \quad r_J < p_1 \leq r_{J+1} \;\&\; r_J < p_2 \leq r_1 + s \;\&\; J < R \\ 1 & \text{if} \quad r_H < p_1 \leq r_{H+1} \;\&\; r_1 < p_2 \leq r_H \;\&\; R < H \\ 0 & \text{if} \quad r_H < p_1 \leq r_{H+1} \;\&\; r_H < p_2 \leq r_R \;\&\; R < H \\ 1 & \text{if} \quad r_H < p_1 \leq r_{H+1} \;\&\; r_R < p_2 \leq r_1 + s \;\&\; R < H \end{cases}$$

where r_1+s is the right limit of C_K

Formally, we can define the save by the following condition

$$\exists\, R:\ n_R < n_1, \quad \text{i.e. } \exists\, R:\ Z_R(p_1,p_2) < Z_1(p_1,p_2)$$

where the index 1 is used to refer to the original grid.

Using this definition we can introduce the save function (Q_R), that expresses the number of accesses to mass memory we save if we choose the R-th partition rather than the original:

$$Q_R = \begin{cases} 0 & \text{if} \quad n_1 \leq n_R \\ n_1 - n_R & \text{if} \quad n_1 > n_R \end{cases}$$

And we can derive that:

$$Q_R(p_1,p_2) = \begin{cases} 1 & \text{if } r_J < p_1 \le r_{J+1} \ \&\ r_1 \le p_2 \le r_J \ \&\ J \ge 2 \\ 0 & \text{otherwise} \end{cases}$$

Then the save function results $Q = \max_{1 \le R \le K} Q_R$

Now if we want an estimation of the average save, we have to evaluate the probability that each point belongs to those sets whose combination gives a contribute different from zero to the save-function.

Denoting as q_R the probability that p belongs to C_R, we can derive, by the definition of $Q_R(p_1,p_2)$, that $Q_R=1$ if p_1 belongs to C_R and p_2 belongs to the union of $C_1, \dots, C_{R-1}$. $\left(p_1 \in C_R \ \&\ p_2 \in \bigcup_{i=1}^{R-1} C_i\right)$

Hence, the average save is given by $QM = \sum_{i=1}^{K} q_i \sum_{j=1}^{i} q_j$.

Replacing in the average save expression the results of Lemma 1 we obtain that the average save is given by $QM = (K-1)/2K$ q.e.d.

Theorem 2

If a bidimensional space with uniform keys distribution is considered, the average save it is possible to obtain through the reply and distribution of information in K grids is

$$QM \equiv (L1+L2)\left\{\frac{(K^2-K+1)(K-1)^2}{(6K(K-1))(K^2)} + \frac{(15K^2-19K+2)(K-1)^2}{(48K(K-1))(K^2)} + \frac{(2K-1)(K-1)}{6K^3}\right\} +$$

$$+ |L1-L2|\left\{\frac{(K+1)(K-1)}{12K^3} + \frac{(K-2)(K+1)(K-1)^2}{48K^3(K-1)}\right\}$$

with Lj: query's j-th dimension width, defined as the number of partition elements intersecting the query image mapped on the logical key space.

Sketch of proof:

The proof is similar to that of theorem 1.

Lemma 1 gives an indication about the value of the δ quantity: since we have shown that the maximum save is obtained when the intervals, determined through the intersection between the K grids, are equal, then δ will be given by:

$$\delta_q^{(j)} = s^{(j)}\, q \,/\, K$$

(with $s^{(j)}$: width of image of original grid buckets along the j-th dimension, expressed in the corrispondent measure unit).

This means that the q-th grid will assume as splitting-values the original grid splitting-values plus $\delta_q^{(j)}$

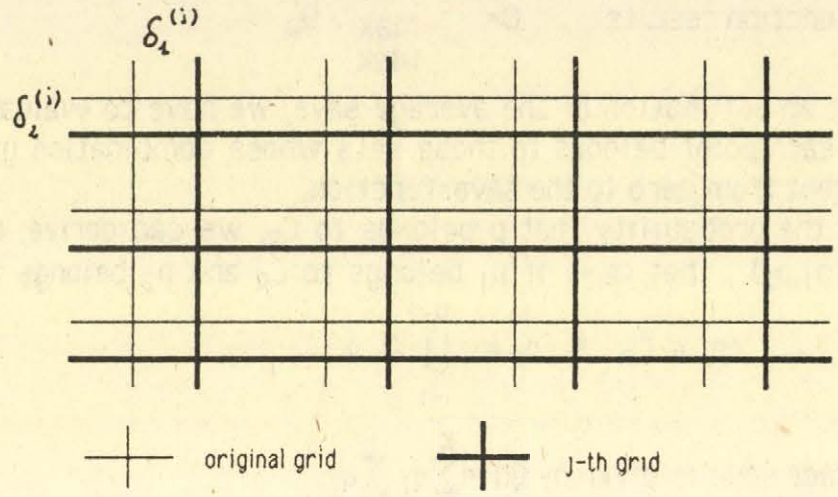


Figura 6

From theorem 2 we obtain, by varying the number of grids and in function of query dimensions L1 and L2:

K=2 --> QM ≅ 0.19 (L1+L2)
K=3 --> QM ≅ 0.27 (L1+L2)
K=5 --> QM ≅ 0.35 (L1+L2)
K=10 --> QM ≅ 0.41 (L1+L2)

$\lim_{k \to \infty}$ QM ≅ 0.48 (L1+L2)

If we want to evaluate the percentual save (QM%), we have to divide QM by the average number of picked buckets, given by (L1*L2).

But in the hypotesis of interactive applications, range queries involve only a few neighbour buckets and (L1+L2) as the same magnitude than (L1*L2).

So the average percent save is around 30% with only three file reproductions, while the asymptotic limit is a little less than 50%.

From this formula we can also see that the behaviour of our structure is better or equal to the original grid-file, but the average save decreases if $(L1+L2) \ll (L1*L2)$.

In other word, in absence of the hypotesis of interactivity, we expect that the data structures tend to have the same behaviour.

To show this we use the following lemma, whose proof is in [2], and theorem 3.

Lemma 2

The average width of a query edge is given by the evaluation of the average distance between two points that are stochastically arranged on an interval with uniform distribution and, if the length of the interval is equal to N, such average width is given by N/2.

Theorem 3

The expression that determines the medium percent save if we do not make any hypothesis on the dimension of the queries is given by

$$QM\% \equiv 4/N \left\{ \frac{(23K^2-27K+10)(K-1)^2}{48K^3(K-1)} + \frac{(2K-1)(K-1)}{6K^3} \right\}$$

Sketch of proof:

It derives from theorem 2 and lemma 2 above, and by assuming $N_1=N_2$, where N_i is the number of splits along the i-th dimension.

From such formula we can derive that, in presence of applications that involve arbitrarily large queries, the medium percent save has order $O(QM\%)=1/N$ with respect to the number of splits in a generic dimension of the space.

4. CONCLUSION

In order to save disk accesses when range or partial match queries are performed, we have introduced a new structure derived from the grid-file through a finite number of duplications and a different distribution of the information in the buckets.

The results, obtained through the analysis presented in the previous chapters, have shown that the new data structure presents its best behaviour when it is applied to interactive environments, while the average save increases with the number of the grids replied.

On the other side, as we expected, at the increase of the size of the query, the new data structure tends, asymptotically, to behave like the original grid-file, even if its behaviour remains always better or equal than that of the grid-file.

An interesting open problem introduced by the new data structure derives from the fact that replies of information require a great amount of disk space, so that it remains to verify whether it is possible to save mass memory space without losing the obtained benefits.

ACKNOWLEDGMENTS

I wish to thank Maurizio Talamo and Giorgio Gambosi for useful suggestions for improvements.

BIBLIOGRAPHY

1) G.Ausiello, G.Gambosi, E.Nardelli, M.Talamo:"GEODEQ: UNO STRUMENTO INTERATTIVO PER LA GESTIONE E L'INTERROGAZIONE DI DATI ED IMMAGINI CARTOGRAFICHE INTEGRATI".AICA Conference, Palermo, 1986.

2) D.Cortolezzis:"DEFINIZIONE DI STRUTTURE DATI DERIVATE DA GRID-FILE PER GESTIONE EFFICIENTE DI DATI MULTIDIMENSIONALI". Thesis, University of Udine, 1987.

3) C.Gaibisso, G.Gambosi, E.Nardelli, G.Soccodato & M.Talamo:"A PROPOSAL FOR THE EFFICIENT REPRESENTATION AND MANAGEMENT OF GEOMETRICS ENTITIES IN A GEOGRAPHIC INFORMATION SYSTEM". Th.Report, University of Rome, 1987.

4) H.Gueting, H.P.Kriegel: "MULTIDIMENSIONAL-B-TREE: AN EFFICIENT DYNAMIC FILE STRUCTURE FOR EXACT-MATCH QUERIES". Th.Report, University of Dortmund, 1985.

5) K.Hinrichs:"IMPLEMENTATION OF THE GRID-FILE: DESIGN CONCEPTS AND EXPERIENCE". BIT n.25, 1985.

6) K.Hinrichs, J.Nievergelt:"THE GRID-FILE: A DATA STRUCTURE DESIGNED TO SUPPORT PROXIMITY QUERIES ON SPATIAL OBJECTS". Intern. Workshop on Graph Teoretic Concepts, Linz 1983.

7) J.Nievergelt, H.Hinterberger, K.C.Sevcik: "THE GRID-FILE: AN ADAPTABLE SYMMETRIC MULTIKEY FILE STRUCTURE". ACM-TODS, vol.9, n.1, 1984.

8) M.H.Overmars: "THE DESIGN OF DYNAMIC DATA STRUCTURES". Lecture Notes in Computer Science, vol. VII, Springer Verlag, 1983.

9) M.Regnier: "ANALYSIS OF GRID-FILE ALGORITHMS". BIT n.25, 1985.

10) N.Roussopoulos, D.Leifker "DIRECT SPATIAL SEARCH ON PICTORIAL DAATBASE USING PACKED R-TREES". ACM SIGMOD 1985.

11) H.Samet: "THE QUAD-TREE AND RELATED HIERARCHICAL DATA STRUCTURE". Computer Surveyes, vol.16, n.2, 1984.

12) H.W.Six, P.Widmayer:"SPATIAL SEARCHING IN GEOMETRIC DATABASES". Th.Report, University of Karlsruhe, 1987.

IMYCS'88, Smolenice Castle, November 14-18, 1988.

PARALLEL CONFLICT-FREE ACCESS TO EXTENDED BINARY TREES

REINER CREUTZBURG

Academy of Sciences of the G.D.R.
Central Institute of Cybernetics
and Information Processes
International Basic Laboratory for
Image Processing and Computer Graphics
Kurstrasse 33, P.O.Box 1298

DDR - 1086 Berlin

1. INTRODUCTION

In general parallel memory schemes are designed for obtaining conflict-free access to arrangements of cells that belong to a specified set of data templates. In case of a single template T the smallest number of memory modules needed obviously is $N = |T|$ (the size of T), but more modules may be required. In general the problem arises of finding the smallest number of memory modules for storing data so as to have conflict-free access for a set of templates of interest.

A lot of research work has been done in designing parallel memories to access arrays and array-like data structures [1,2,5,7,9-11,13,15,17]. Trees are another important data structure in computer science [8]. It is an interesting problem to design parallel memories to access trees or tree-like data structures [3-6,12,14-17]. In recent papers [3,4] we have

investigated the parallel access to complete subtrees of trees.

The aim of this paper is to investigate the parallel conflict-free access to complete extended binary subtrees of binary trees.

The extended binary tree [8] is the fundamental data structure in modern logic programming languages, like LISP and PROLOG. Therefore the results of this paper are assumed to be of significant interest for the design of specialized hardware structures for future parallel artificial intelligence machines.

Consider a labelled binary tree such that the immediate successors of the node x are 2x+1, 2x+2, and the label of the root is 0. The level of a node is defined by initially letting the root be at level 1. The level of every node is one more than the level of its immediate predecessor. The height t of a tree is defined as the maximum level of any node in the tree.

Parallel access to extended binary trees means the conflict-free access to all the 2t - 1 nodes of a complete extended binary subtree of height t with an arbitrary node x as root, as shown in figure 1. We consider left-, right- and general-extended binary trees.

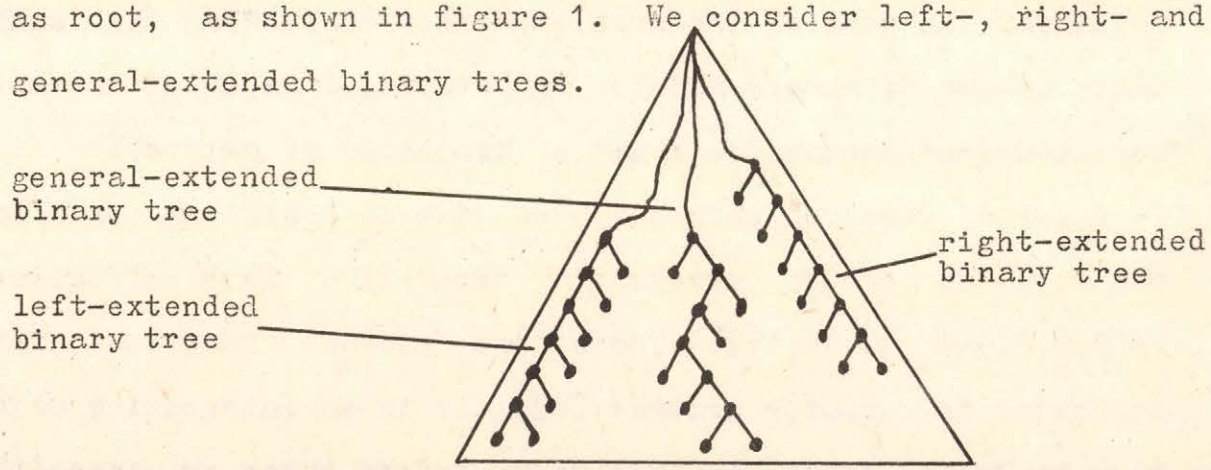


Fig.1 Labelled complete extended binary subtrees of height 7 with 13 nodes in a complete binary tree

By definition a binary tree is called a complete extended binary tree if it contains exactly two nodes in each level, exactly one of them is a leaf (except of the root level which contains no leaf and the last level which contains two leafs).

2. RECURSIVELY LINEAR MODULE ASSIGNMENT FUNCTIONS

A memory module assignment function S is a mapping from the set of the labels of an extended binary tree to the N memory modules. We denote the set of indices of memory modules by

$$E_N = \{0,1,\ldots,N-1\} .$$

A recursively linear module assignment function S for binary trees is given by the following recurrence equations

$$\begin{aligned} S(0) &= 0 \\ S(2x+1) &= a\,S(x) + b \quad \bmod N, \\ S(2x+2) &= c\,S(x) + d \end{aligned} \tag{1}$$

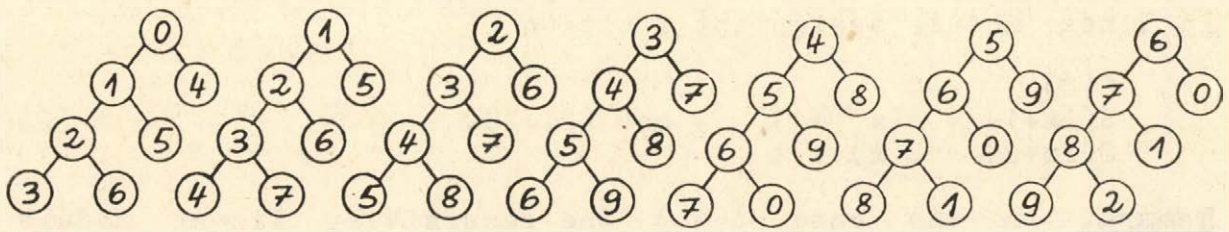
where a,b,c,d are integers with $0 < a,b,c,d < N$.

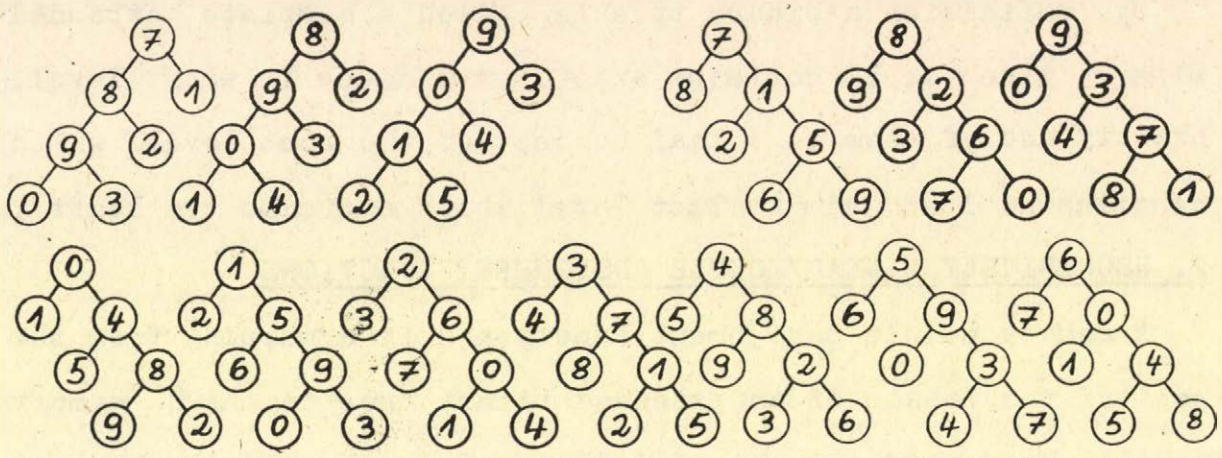
The following example is illustrative.

Example 1. The special recursively linear module assignment function S with

$$\begin{aligned} S(0) &= 0 \\ S(2x+1) &= S(x) + 1 \quad \bmod 10 \\ S(2x+2) &= S(x) + 4 \end{aligned}$$

allows the conflict-free access to all 7 nodes of an arbitrary complete left- or right-extended binary subtree of height 4. The relating 20 subtrees are the following:





Theorem 1. A parallel conflict-free access to all the $2t - 1$ nodes of an arbitrary complete left-extended binary subtree of height t of a complete binary tree is possible with $N = 2t - 1$ memory modules using the recursively linear module assignment function S

$$\begin{aligned} S(0) &= 0 \\ S(2x+1) &= S(x) + 1 \qquad \mod (2t-1). \\ S(2x+2) &= S(x) + t \end{aligned}$$

The **proof** is given in an extended version of this paper [16].

A similar result [16] can be given for complete right-extended subtrees by simply exchanging the left- and right successor function parts in (3).

Theorem 2. A parallel conflict-free access to all the $2t - 1$ nodes of an arbitrary complete left- or right-extended binary subtree of height t $(t > 3)$ of a complete binary tree is possible with $N = 2t + 2$ memory modules using the recursively linear module assignment function S

$$\begin{aligned} S(0) &= 0 \\ S(2x+1) &= S(x) + 1 \qquad \mod (2t+2). \\ S(2x+2) &= S(x) + t \end{aligned} \tag{2}$$

Remark. In the case $t = 3$ the recursively linear module assignment function (5) can be taken modulo $N = 2t + 1 = 7$.

The **proof** is given in an extended version [16] of this paper.

3. RECURSIVELY NONLINEAR MODULE ASSIGNMENT FUNCTIONS

Although the recursively linear module assignment function S according to (2)

- is easy to implement in hardware,
- needs only three more memory modules than accessed nodes,
- and allows the conflict-free access to complete left- and right-extended binary subtrees,

it does not allow the conflict-free access to general-extended binary subtrees, in general. The following theorem shows how the number of memory modules can be further reduced and complete general-extended binary subtree access is possible by use of a nonlinear module assignment function.

Theorem 4. A parallel conflict-free access to all the $2t$ - nodes of an arbitrary complete left-, right- or general-extended binary subtree of height t ($t > 2$) of a complete binary tree is possible with $N = 2t$ memory modules using the recursively (nonlinear) module assignment function S given by $S(0) = 0 \bmod N$ and the table

$S(x)$	0	1 2	3 4	5 6	7 8	...	2t-3 2t-2	2t-1
$S(2x+1)$	1	3 3	5 5	7 7	9 9	...	2t-1 2t-1	1
$S(2x+2)$	2	4 4	6 6	8 8	10 10	...	0 0	2.

The proof is shortly illustrated by the following figure 2. and is given in an extended version [16] of this paper. Starting with $S(0) = 0 \bmod N$ we obtain the structure of the complete binary tree in figure 2. Then obviously every complete left-, right- or general binary extended subtree access is possible. Two examples are marked with bold lines.

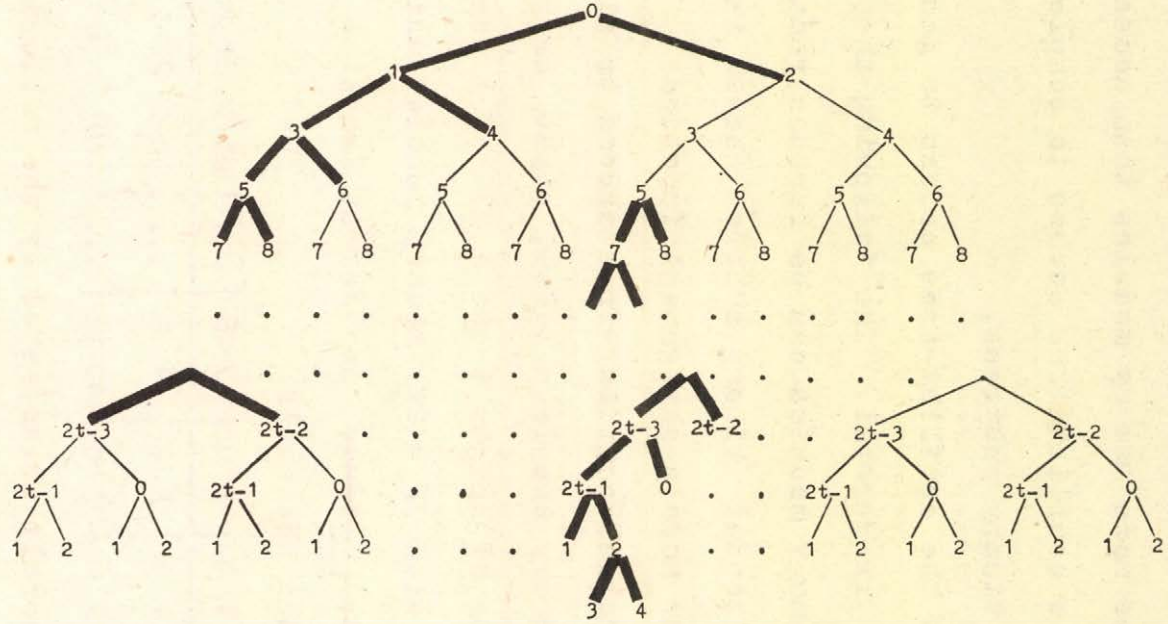


Fig.2

REFERENCES

[1] Batcher, K. E.: STARAN parallel processor system hardware. Proc. Fall Joint Computer Conf. AFIPS Conf., AFIPS Press, 43, 1974, pp.405-410

[2] Budnik,P., and D.J.Kuck: The organization and use of parallel memories. IEEE Trans. Comput. C-20 (1971), pp.1566-1569

[3] Creutzburg, R.: Parallel optimal subtree access with recursively linear memory function. Proc. PARCELLA'86 Berlin, (Eds.: T. Legendi, D. Parkinson, R. Vollmar, G. Wolf) Akademie-Verlag: Berlin 1986, pp.203-209

[4] Creutzburg, R.: Parallel linear conflict-free subtree access. Proc. Internat. Workshop Parallel Algorithms Architectures (Suhl 1987), Akademie-Verlag Berlin 1987, (Eds.: A. Albrecht, H. Jung, K. Mehlhorn) pp.89-96

[5] Gößsel, M., and B. Rebel: Data structures and parallel memories. Proc. PARCELLA'86 Berlin, Akademie-Verlag Berlin 1986, (Eds.: T.Legendi, D. Parkinson, R.Vollmar, G.Wolf), pp.49-60

[6] Gößsel, M., and B. Rebel: Memories for parallel subtree access. Proc. Internat. Workshop Parallel Algorithms Architectures (Suhl 1987), Akademie-Verlag Berlin 1987, (Eds.: A. Albrecht, H. Jung, K. Mehlhorn), pp.122-130

[7] Hockney, R. W., and C. R. Jesshope: Parallel Computers. Hilger: Bristol 1981

[8] Knuth, D. E.: The Art of Computer Programming, Fundamental Algorithms. Addison-Wesley: Reading (MA) 1968

[9] Kuck, D. J., and R. A. Stokes: The Burroughs scientific processor. IEEE Trans. Comput. C-31 (1982), pp. 363-376

[10] Lawrie, D. H., and Ch. R. Vora: The prime memory system for array access. IEEE Trans. Comput. C-31 (1982), pp.435-442

[11] Shapiro, H. D.: Theoretical limitations on the use of parallel memories. Univ. Illinois, Dept. Comp. Sci., Rep. No. 75-776 Dec. 1975

[12] Shirakawa, H.: On a parallel memory to access trees. (Unpublished) Ritsumeikan Univ., Kyoto, Japan, 1984

[13] Wijshoff, H. A. G., and J. van Leeuwen: The structure of periodic storage schemes for parallel memories. IEEE Trans. Comput. C-34 (1985), pp.501-505

[14] Wijshoff, H. A. G.: Storing trees into parallel memories. Proc. 1985 Int. Conf. Parallel Computing, (Eds.: M. Feilmeier, J. Joubert, U. Schendel, Elsevier: Amsterdam 1986), pp.253-261

[15] Wijshoff, H. A. G.: Data organization in parallel computers. Diss. (Rijksuniv. Utrecht, Netherlands), 1987

[16] Creutzburg, R.: Parallel conflict-free access to extended binary trees. Preprint ZKI Berlin (1988)

[17] Gößsel, M.; B. Rebel, and R. Creutzburg: Memory architecture and Parallel Access. Akademie-Verlag: Berlin (in print)

IMYCS'88, Smolenice Castle, November 14-18, 1988.

A PARTIALLY PERSISTENT DATA STRUCTURE FOR THE SET-UNION PROBLEM WITH BACKTRACKING

CARLO GAIBISSO - Istituto di Analisi dei Sistemi ed Informatica del CNR, Viale Manzoni 30, 00185 Roma, Italy.

ABSTRACT: An extension of the well known Set-Union problem is considered, where searching in the history of the partition and backtracking over the Union operations are possible. A partially persistent data structure is presented which maintains a partition of an n - item set and performs each Union, each Find and each search in the past in $O(\lg n)$ time per operation, at the same time allowing to backtrack over the sequence of Unions in constant time. The space complexity of such a structure is $O(n)$.

1. INTRODUCTION

The Set - Union problem and its variants has been extensively studied in recent years [1, 2, 3, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17].

The original problem was [10] to mantain a representation of a partition of a set $S = \{ 1, 2, \ldots, n \}$ under the following two operations:

Union (X, Y, Z): return a new partition of S in which subsets X and Y are merged into one subset $Z = X \cup Y$;

Find (x) : given an item $x \in S$, return the name of the (unique) subset containing x.

Initially, each element is assumed to be a singleton.

A first naive solution. proposed by Galler and Fischer [8] . requires O(1) time per Union and O(n) time per Find in the worst case.

This bound has been remarkably improved by the use of techniques of balanced linking (link by rank, link by size) [8, 15], which made it possible to derive solutions requiring O(1) time per Union and O ($lg\ n$) time per Find in the worst case.

The best solution with this type of approach is due to Blum [2] and requires O($lg\ n / lglg\ n$) single operation worst -case time complexity.

After the introducing of the path compression technique [1], the problem has been extensively studied from the point of view of worst-case time on sequences of Union and Find operations [3,10,15], i.e. of the amortized complexity of operations [14].

The best solution in this direction has been given in [15]: it requires O(n) space and O($m\,\alpha(m + n, n) + n$) running time, where m is the number of Find operation performed and $\alpha(\ldots,\ldots)$ is a very slowly increasing function related to the inverse of Ackermann's function. In such a paper it was also proved that such a complexity is also a lower bound for a very general class of algorithms.

As an extension Mannila and Ukkonen [11] proposed a variant of this problem, relevant in the framework of logic programming interpreters design, introducing a new operation defined as follows:

Deunion: undo the last Union performed, i.e. return to the state immediately preceding the execution of such Union.

In such a paper, an algorithm is presented which is efficient with respect to amortized time complexity.

Successively, Westbrook and Tarjan [17] completely characterized such problem giving a $lg\ n\ /\ lglg\ n$ upper bound of the amortized complexity and proving that such bound is also a lower bound for the class of separable algorithms.

Gambosi, Italiano and Talamo [9] also considered a generalization of such problem to the case where a (real) weight is associated to each Union and the Deunion operation is substituted by:

Backtrack: returns to the state immediately before the execution of the Union of largest weight thus far performed.

The solution presented in [9] makes it possible to perform Unions in $O(\lg\lg n)$, Finds in $O(\lg n)$ and Backtracks in O(1) time per operation, using a data structure with $O(n)$ space complexity.

Gaibisso, Gambosi and Talamo [7] introduced a partially persistent [4] version of the classical Set-Union problem which allows a search in the history of the partition. They introduced a new kind of Find, referred to as PFind, defined as follows:

PFind (x, k): given an item $x \in S$, return the name of the (unique) subset containing x just after the k - th Union operation was performed.

Such operation includes the usual Find as a particular case.

The partially persistent data structure presented in such a paper mantains a partition of an n - item set performing each Union in O(1) and each PFind in $O(\lg n)$ worst case time, while the space complexity is $O(n)$.

In this paper a further extension of the Set-Union problem is considered where both searching in the history of the partition and backtracking over the sequence of Union operations are possible.

Motivations for the study of the Union-PFind-Backtrack problem may, for example, arise from the implementation of search heuristics in the framework of Prolog environment design.

The main results of the paper are concerned with the worst-case per operation analysis of the Union, PFind and Backtrack operations: it is shown how to perform each Backtrack in O(1), each Union and PFind in $O(\lg n)$ worst-case times, using a partially persistent data structure which requires $O(n)$ space complexity.

The remainder of this paper is organized as follows. In section 2 a data structure for the Union-PFind-Backtrack problem is introduced. in section 3 its worst-case time and space complexity are anlysed. Section 4 contains some concluding remarks.

2. THE DATA STRUCTURE.

Recalling the concepts introduced in the last section, the problem considered is that of maintaining a representation of a partition of a set $S = \{ 1, 2, ..., n \}$ under the following three operations:

Union (X, Y, Z) : return a new partition of S in which subsets X and Y are merged into one subset $Z = X \cup Y$;

PFind (x,k) : given an item $x \in S$, return the name of the (unique) subset containing x just after the k - *th* Union operation was performed;

Backtrack(k) : return to the state just after *the* k - *th* Union operation was performed.

In the sequel a Union will be referred to as "valid" if it has not been undone by backtracking and as "void" otherwise.

It is not difficult to see that at any time the actual partition is the same that would have been resulted from simply applying the currently valid Unions to the initial set of singletons, in exactly the same order in which such Unions were performed in the actual sequence: this individuate a virtual sequence, referred to as VS, of valid Unions. At any time is henceforth possible to univocally denote each valid Union by the ordinal number it gets in that virtual sequence.

Furthermore it can be proved that each Union, as long as it remains valid, maintains the same ordinal number it was given at the time of its execution.

Since if the parameter k exceeds the number n' of operations currently in VS, then it cannot refer to void Unions: it follows that PFind(k) and Backtack(k), for $k \geq n'$, reduce respectively to PFind (n') and Backtrack (n').

As a consequence, it is worth noting that once a Backtrack(k) has been performed the k - *th* Union in VS can be considered the last Union performed.

A data structure able to support the Union, Find, PFind and Backtrack operations is now introduced: as in most of the algorithms which deal with versions of the Set-Union problem, each set of the partition is maintained as a tree whose root contains the name of the set. When a Union is performed, exactly one link between two nodes is introduced, which is associated to such an operation: recalling the previously given definitions of valid and void Union, also a link is said to be valid or void according to its corresponding Union and hence valid links return a connection which has not yet been cancelled by backtracking.

In order to support the PFind and Backtrack operations such a structure has been modified in the following way:

- if a link has been introduced by the i-*th* Union operation performed, such a link is marked with the integer i. The mark associated to a link l will be referred to as Mark (l);
- each node p in the data structure has an associated dictionary Dictionary(p), such that each item in Dictionary(p) corresponds to a link l entering p and stores the name and the rank associated to such a node after the Union introducing l was performed. Initially, each node p has an associated dictionary whith only one item (0,0, n).
- each link in the data structure is directly accessible by means of an (n-1)-item array "Access".Two indices "i_{valid}" and "i_{max}" allow insertions in the array in such a way that Access [i] , $1 \leq i \leq i_{valid}$, points to the link introduced by the i - th valid Union in the virtual sequence of Unions performed, while Access [j], $i_{valid} + 1 \leq j \leq i_{max}$, points to a void link, if it exists. In other words i_{valid} and i_{max} point respectively to the last valid and the last void links inserted. It is worth noting that in such a way the validity of a link l can be tested in constant time by simply comparing Mark(l) with i_{valid}. Initially $i_{valid} = i_{max} = 0$.

In the sequel Root(X) will refer to the root of the tree representing subset X in the data structure.

The different operations can now be implemented as follows:

Union (X, Y, Z): first of all if $i_{valid} < i_{max}$ then at least one void link is present in the structure. In such a case consider the void link l from node p' to node p pointed to by Access[i_{valid} + 1] and delete link l from the data structure and the node corresponding to l from Dictionary(p).

For what concerns the rank and the name associated to p before l was introduced in the data structure, it is worth noting that they can be retrieved in Dictionary(p) stored in the item corresponding to the live link entering p most recently added, i.e. in the node which stores the maximum mark less than i_{valid}.

Set, in any case, i_{valid} to i_{valid} + 1 and i_{max} to the maximum between i_{valid} and i_{max} itself; enter a new link (Root(X), Root(Y)) or (Root(X), Root(Y)) according to the linking by rank strategy and store a pointer to such a link in Access [i_{valid}].

Finally add a new item (i_{valid}, Z, r) in Dictionary (Root (Y)), where r is the new rank of Root (Y) after the Union operation has been performed.

Figures 2.1a and 2.1b show how the structure evolves when a Union operation is performed.

PFind(x,k) : starting from node x , which belong to some tree T, traverse the path from this node to Root (T) until:

1) a node p is reached such that either p = Root (T) or
2) for the (unique) link l outgoing from p the condition Mark(l) > k holds (since the parameter k cannot refer to void Unions the above mentioned condition also tests link validity).
   Let D = { x | x = Mark (l), l enters p, l is a valid link , $x \leq k$ }, then return the name stored by the node corresponding to the link L' = *max* D in Dictionary (p).

Backtrack(k) : set i_{valid} to k.

3. WORST-CASE TIME AND SPACE COMPLEXITY ANALISYS.

As far the worst-case time and space complexity are concerned, the following theorem can be stated:

Theorem:

with the previously introduced structure it is possible to perform:

a) the Union operation in O($lg\ n$) worst case time;

b) the PFind operation in O($lg\ n$) worst case time;

c) the Backtrack operation in O(1) worst case time;

d) the amount of the space to store the data structure is $S(n) = O(n)$.

Let us first note that it is possible to easily implement each dictionary in such a way that inserting an item, deleting an item and searching for an item all require $O(lg\ m)$ worst case time, where m is the number of items in the dictionary . Furthermore, since the number of links, void or valid, in the data structure cannot exceed $n - 1$, one for each item in the array Access , the sum of items in the dictionaries, and consequently the number of items in a single dictionary, is $O(n)$.

It is now possible to prove the theorem.

Proof:

a) implementing each Union operation requires a constant number of insert, delete, and search operations onto the dictionaries associated to the data structure nodes: each one of such operations requires O(lg n) time. Moreover a constant number of updates on the data structure and the indices are performed, which require constant time. Thus each Union operation can be performed in O($lg\ n$) worst case time;

b) each PFind(x, k) implies two searches in the structure:

the first search, starting from node x to locate a node p such that either p is the root of the tree containing x, or for its outgoing link l the condition Mark(l) > k holds. This search obviously takes O($lg\ n$) time.

The other search is performed in Dictionary(p) in order to access the name of the subset containing item x just after the k - th Union operation was performed: also this second search takes O ($lg\ n$) time.

Hence each *PFind* operation takes O($log\ n$) time;

c) trivial;

d) since the number of the nodes in the trees representing the whole partition, the number of items in the dictionaries and the array dimension all are $O(n)$, the space complexity of the whole data structure is obviously $O(n)$.

4. CONCLUSIONS

In this paper an extension of the Set-Union problem has been considered, where searching in the history of the partition and backtracking over the Union operations are possible.

A partially persistent data structure which support each Union and each search in the past in $O(\lg n)$ time per operation and allows backtracking in O(1) time, has been proposed. The space required to implement such a structure is $O(n)$.

REFERENCES

[1] A.V. Aho, J.E.Hopcroft, J.D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison - Wesley (1974).

[2] N.Blum. *On the single Operation Worst -case Time Complexity of the Disjoint Set Union problem*, Proc. 2nd Symp. on Theoretical Aspects of Computer Science (1985).

[3] B.Bollobas, I.Simon, *On the expected Behavior of Disjoint Set Union Algorithms*, Proc. 17th ACM Symp. on Theory of Computing (1985).

[4] J.R. Driscoll, N. Sanark, D.D. Sleator, R.E. Tarjan, *Making Data Structures Persistent*, Proc. 18th Symp. on Theory of Computing STOC (1986).

[5] M.J. Fischer, *Efficiency of Equivalence Algorithms*, in Complexity of Computations, R.E. Miller and J.W. Thatcher, eds., Plenum Press, New York (1972).

[6] H.N. Gabow, R.E. Tarjan, *A Linear Time Algorithm for a Special case of Disjoint Set Union*, Proc. 15th ACM Symp. on Theory of Computing (1983).

[7] C.Gaibisso, G.Gambosi, M.Talamo, *A Partially Persistent Data Structure for the Set-Union Problem* submitted to RAIRO Theoretical Informatics and Apllications, (1987).

[8] B.A. Galler, M.J. Fischer, *An Improved Equivalence Algorithm*, Comm. ACM 7 (1964).

[9] G. Gambosi, G.F. Italiano, M. Talamo, *Worst-Case Analysis of the Set Union Problem with Backtracking* to appear on "Theoretical Computer Science" (1988).

[10] J.E.Hopcroft, J.D. Ullman, *Set Merging Algorithms*, SIAM J. Comput. 2 (1973).

[11] H.Mannila, E. Ukkonen, *The Set Union Problem with Backtracking*, Proc. 13th ICALP (1986).

[12] R.E. Tarjan, *Efficiency of a Good but not Linear Disjoint Set Union Algorithm*, J. ACM 22 (1975).

[13] R.E. Tarjan, *A Class of Algorithms which Require Linear Time to Mantain Disjoint Sets*, J. Computer and System Sciences 18 (1979).

[14] R.E. Tarjan, *Amortized Computational Complexity*, SIAM J. Alg. Discr. Meth. 6, (1985).

[15] R.E. Tarjan, J. van Leeuwen, *Worst-Case Analysis of Set Union Algorithms*, J. ACM 31 (1984).

[16] J. van Leeuween, T. van der Weide, *Alternative Path Compression Techniques*, Techn. Rep. RUU-CS-77-3, Rijksuniversiteit Utrecht, The Netherlands.

[17] J.Westbrook, R.E. Tarjan, *Amortized Analysis of Algorithms for Set-Union with Backtracking*, Tech. Rep. TR-103-87, Dept. of Computer Science, Princeton University, (1987).

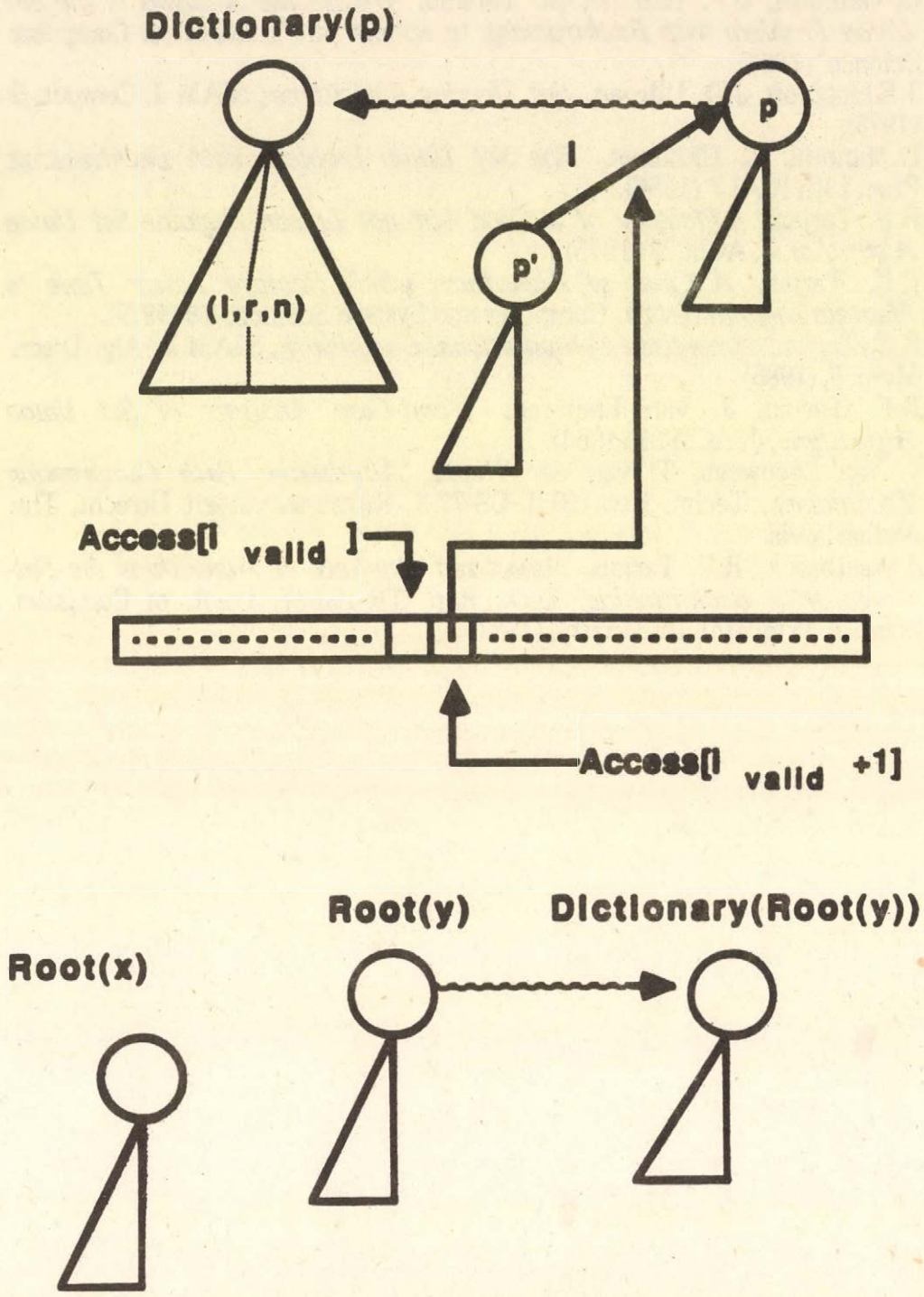

Dictionary(p)
p
l
p'
(l,r,n)
Access[l valid]
Access[l valid +1]
Root(y)
Dictionary(Root(y))
Root(x)

figure 2.1.a

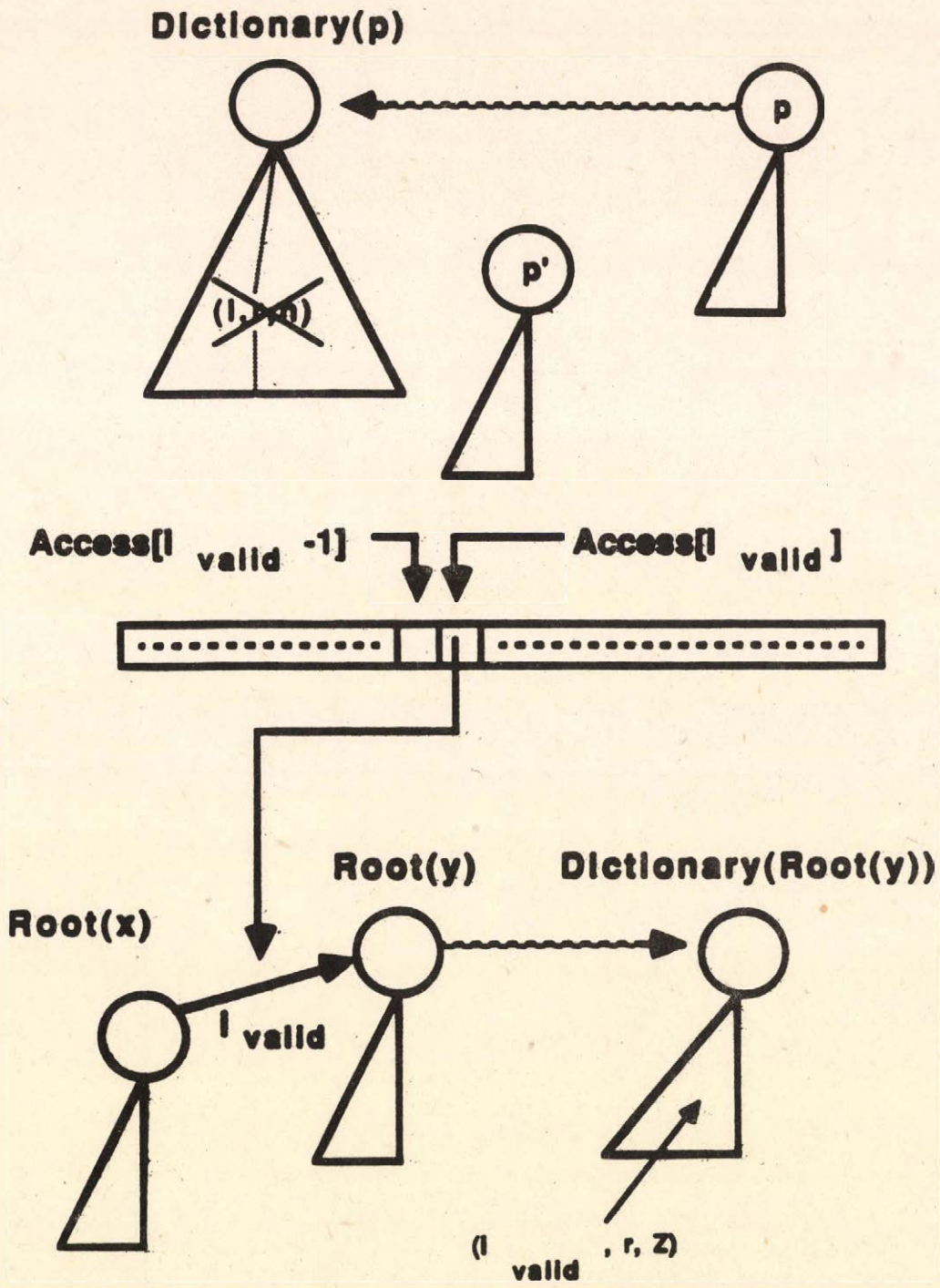
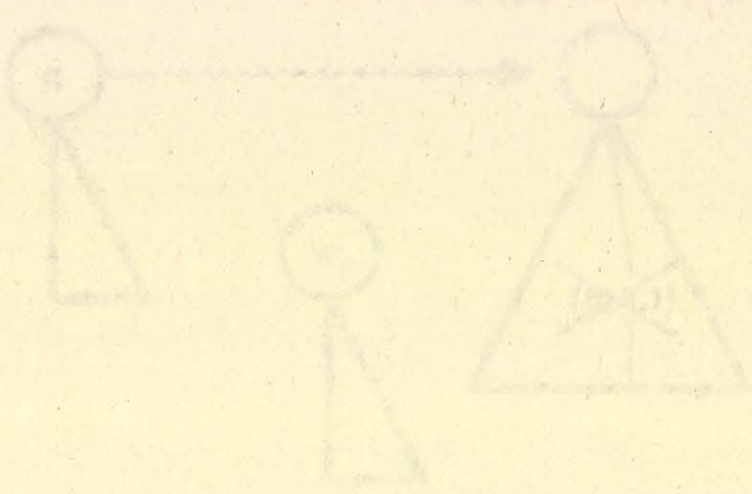

Dictionary(p)
p
p'
Access[l valid -1]
Access[l valid]
Root(y)
Dictionary(Root(y))
Root(x)
l valid
(l valid, r, Z)

figure 2.1.b



IMYCS'88, Smolenice Castle, November 14-18, 1988.

QUESTIONS OF DECIDABILITY FOR CONTEXT-FREE CHAIN CODE PICTURE LANGUAGES

by

FRIEDHELM HINZ

RWTH Aachen, Lehrstuhl Informatik II, Templergrab. 55,
5100 Aachen, Federal Republic of Germany;

research was done during the author's stay at IIG,
Institutes for Information Processing,
Graz University of Technology,
Schiessstattgasse 4a, 8010 Graz, Austria

to the glory of God, who gave wisdom to man

Abstract. A picture description is a word over the alphabet {u,d,r,l,↑,↓} with the following interpretation: "move one unit line up (down, right, left, resp.) from the current point" for letter u (d,r,l, resp.) and "lift (sink) the pen" for ↑ (↓) . The set of unit lines traversed with sunk pen is the picture described. A set of pictures given by a set of picture descriptions generated by a context-free grammar is called a context-free (chain code) picture language.

We show that the membership problem and the superpicture problem are undecidable for context-free and for linear context-free picture languages, while the subpicture problem is decidable. This result is in contrast to the situation for regular picture languages and for context-free picture languages generated without using a "lift the pen"-symbol. For these classes all three problems are known to be decidable, and the subpicture problem is harder than the superpicture problem (NP-complete instead of polynomial).

1. INTRODUCTION

There are various approaches to questions of picture description and pattern recognition that apply the knowledge of formal language and automata theory. Acceptor concepts for array picture languages are investigated in [1]. A close connection between string language theory and picture language theory can be established following a suggestion of [7] to interprete a string as a traversal of a picture through its subpatterns. The concept of chain code introduced in [2] can be viewed as an interesting special case thereof. Basically it allows to traverse a picture in the plane using eight directions as shown in Fig.1.1a. Additional features allow to lift the pen to take it to a new position and sink it again. Subsequent papers simplify the concept using only an alphabet of four letters, {u,d,r,l} , corresponding to the four directions up, down, right, left. For example, the word "urrrddurrldl" describes the picture shown in Fig.1.1b.

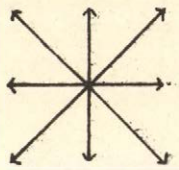


Fig.1.1a

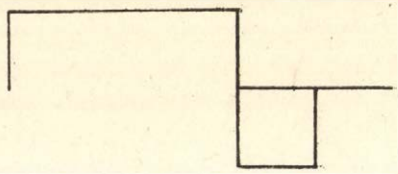


Fig.1.1b

A hierarchy of picture languages analogous to the classical Chomsky hierarchy is established in [4]. The membership problem is shown to be NP-complete by [8] and [3] for regular and context-free chain code picture languages (i.e. sets of pictures described by words in regular resp. context-free string languages). The subpicture problem and the superpicture problem (decide for a picture and a language, whether the picture is subpicture resp. superpicture of a picture described in the language) are shown to be decidable for context-free languages in [4].

The investigation of the class of general picture languages, that allow "invisible lines", i.e. that are generated by gram-

mars over $\{u,d,r,l,\uparrow,\downarrow\}$ is started in [9]. It is proven that the membership and the subpicture problem are NP-complete for regular picture languages both in the restricted and in the general concept, and that the superpicture problem is solvable in polynomial time. For the restricted concept the superpicture problem of context-free picture languages is also solvable in polynomial time by an algorithm in [6], but for the general concept we will show that the superpicture problem is undecidable. Even for a linear context-free picture language the membership and the superpicture problem will turn out to be undecidable. In contrast to this we state without proof here that the subpicture problem is decidable in NP time for all context-free picture languages [10].

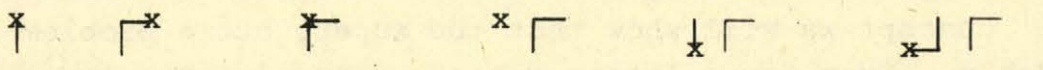
We will first give some basic definitions (Section 2). In Section 3 we simulate a two-counter-machine by a linear grammar and a set of pictures. Thereby we show that the membership and the superpicture problem are undecidable for linear context-free picture languages. Finally (Section 4) we add some remarks concerning possible variants of the description concept.

2. BASIC DEFINITIONS AND ELEMENTARY PROPERTIES

Let Z be the set of integers and N the set of nonnegative integers. We adopt the basic definitions of formal language theory. Definitions about pictures and picture languages will be given informally, rigorous definitions can be found in the literature cited above. An attached picture is a finite set of unit lines in the two-dimensional integer grid. An (unattached) picture is an equivalence class of attached pictures, where two attached pictures are equivalent if they differ only by their relative position in Z^2, but the shape is the same.

The best investigated picture description alphabet so far is the set of unit vectors $\{u,d,r,l\}$, where $u = (0,1)$, $d = (0,-1)$, $r = (1,0)$, $l = (0,-1)$. However, it can be used to describe connected pictures only. To describe possibly disconnected pictures one may join the set of states $\{\uparrow,\downarrow\}$ to the

alphabet, where ↑ indicates lifting the pen and ↓ indicates sinking the pen. A state-letter occurring in a word specifies the state of the sequence of moves indicated by the subsequent vector-letters. For example in the word "ur↑ll↓dl" the moves indicated by the subword "ll" are executed with lifted pen and the other with sunk pen, see Fig.2.1.

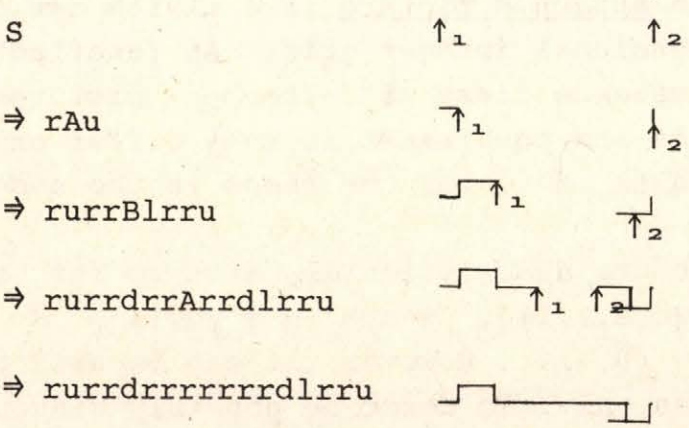


u ur ur↑l ur↑ll ur↑ll↓d ur↑ll↓dl

Fig.2.1 "x" indicates the current position of the pen

The picture described by a word $w \in \{u,d,r,l,\uparrow,\downarrow\}^*$ is denoted by pic(w) . Any grammar G (regular, linear, context-free, etc.) generates its string language L(G) and the corresponding picture language $Pic(G) := \{pic(w) \mid w \in L(G)\}$ (regular, linear, context-free, etc.). We define the language of all superpictures with respect to G as $Super(G) := \{p' \mid p'$ is superpicture of some $p \in Pic(G)\}$ and the language of all subpictures as $Sub(G) := \{p' \mid p'$ is subpicture of some $p \in Pic(G)\}$.

Intuitively speaking, drawing a picture according to a derivation in a linear grammar can be done using two cursors: The first one works according to the letters left of the variable and the second one according to the letters right of the variable, in opposite direction. For an example see the following figure:



3. SIMULATING A TWO-COUNTER-MACHINE

In this section we introduce the notion of a two-counter-machine. From [5] we know how to simulate a Turing machine by a two-counter-machine. We present an idea to simulate a two-counter-machine by a linear grammar and a set of pictures. This allows to transfer results about Turing machines to linear picture languages, specifically we obtain that the membership problem and the superpicture problem are undecidable.

A two-counter-machine (tcm) works with two integer counters and a finite memory. Each of the counters may be incremented or decremented or tested for zero, so the set of elementary commands is $C = \{inc1, inc2, dec1, dec2, if1, if2\}$. To be precise, C is the set of partial functions from Z^2 to Z^2 given by $inc1(m,n)=(m+1,n)$, $dec1(m,n)=(m-1,n)$, $if1(0,n)=(0,n)$ and $if1(m,n)$ is undefined for $m \neq 0$; the other commands work analogously on the second component. A tcm is a tuple $M = (Q,q_0,\delta,F)$, where Q is a finite set of control states, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of final states, and the transition relation δ is a subset of $QXCXQ$. Intuitively speaking, a triple $(q,c,q') \in \delta$ indicates that for control state q the tcm may carry out command c and change control to the state q' . We define the mapping of M as $f_M: Z^2 \to Pot(Z^2)$ given by $f_M(m,n) = \{c_t \circ \ldots \circ c_1((m,n)) \mid$ there is a word $(q_0,c_1,q_1)\ (q_1,c_2,q_2) \ldots\ (q_{t-1},c_t,q_t) \in \delta^*$ with $q_t \in F\}$. For example to multiply with 2 we need a machine DUP with $f_{DUP}(m,0) = \{(2m,0)\}$ for every $m \in N$. A solution of this problem is the tcm $DUP = (\{q_0,q_1,\ldots,q_5\},q_0,\delta,\{q_5\})$ with δ given by

$(q_0,dec1,q_1)$ }
$(q_1,inc2,q_2)$ } shift the value of the first counter to the
$(q_2,inc2,q_0)$ } second and duplicate it
$(q_0,if1,q_3)$ } assure that the above operation is finished
$(q_3,dec2,q_4)$ } shift the value of the second counter to
$(q_4,inc1,q_3)$ } the first one
$(q_3,if2,q_5)$ } assure that the above operation is finished

In an analogous way one can construct a machine that multiplies with an arbitrary fixed integer instead of 2 or divides by an arbitrary fixed integer. Combining such multiplication machines and division machines Minski [5] builds a tcm to simulate a Turing machine. The concept of Minski is different from our concept, as he allows nonnegative integers for the counters only, i.e. a dec-command is defined for positive value of its counter only. Note that with this change of meaning we still have $f_{DUP}(m,0)=\{(2m,0)\}$ for every $m\in N$. The functions computed by the multiplication machines and division machines of [5] are not affected within this relevant range either. Therefore, we can carry over [5, Theorem 1] to our machine concept and obtain the following

3.1 Proposition Every Turing machine T over the alphabet $\{0,1\}$ is represented by a tcm M in the following sense: If and only if T , started at the x-th square of its tape with the binary number k as input, may reach a final state, then $f_M((2^k3^{2^x},0))$ is nonempty.

3.2 Theorem For every tcm M there is a linear grammar G such that for every $m\in N$ the following assertions are equivalent:

(i) $f_M(m,0)\neq\emptyset$
(ii) $pic(d\uparrow r^m\downarrow r)\in Pic(G)$
(iii) $pic(d\uparrow r^m\downarrow r)\in Super(G)$

Proof An arbitrary tcm can easily be transformed equivalently such that it stops with empty counters only. So for $f_M(m,0)\neq\emptyset$ we assume w. l. o. g. $f_M(m,0)=\{(0,0)\}$. Let $p=pic(d\uparrow r^m\downarrow r)$. The vertical line in p is called the zero-mark and will be drawn by the grammar G using the word $z:=\downarrow ud$ as often as there is a test for zero in a corresponding computation. The horizontal line is called the input mark and will be drawn only in the first step. Recall that drawing a picture according to a derivation in a linear grammar can be understood as drawing it with two cursors. The main idea of the proof is to represent the values of the counters of M as positions of

the cursors relative to the zero-mark and will be explained in the following. For $M=(Q,q_o,\delta,F)$ we choose a new start symbol S together with the set Q as variables for G and the following set of productions:

$S \to lq_oz$ (initiatory rule)

$q \to \uparrow rq'$ for $(q,inc1,q')\in\delta$ $\quad q \to q'\uparrow l$ for $(q,inc2,q')\in\delta$

$q \to \uparrow lq'$ for $(q,dec1,q')\in\delta$ $\quad q \to q'\uparrow r$ for $(q,dec2,q')\in\delta$

$q \to zq'$ for $(q,if1,q')\in\delta$ $\quad q \to q'z$ for $(q,if2,q')\in\delta$

$q \to \lambda$ for $q\in F$ (final rules)

Given a successful computation of M on input $(m,0)$ we construct a derivation of G as follows: In the first step apply the initiatory rule, in the $(i+1)$-st step apply the rule corresponding to the i-th step of the computation for $i\in\{1,\ldots,n\}$, (where n is the length of the computation,) and in the $(n+2)$-nd step finish with a rule of the form $q\to\lambda$. Since the computation ends in a final state, a suitable rule of the form $q\to\lambda$ must exist. A final rule of this form means that both cursors used to draw the picture according to the derivation meet in the same vertex in the last step. Define this vertex as zero and remember that M accepts with value zero in both counters. Then it is easily seen by induction that in every step the values of the counters are the same as the positions of the cursors. This has two consequences: On the one hand the first step of the derivation will leave the cursors in positions m and 0, so by definition of the initiatory rule in the first step the picture p is drawn. On the other hand a rule of the form $q\to zq'$ can be applied only when the first cursor is at vertex zero, since a triple $(q,if1,q')$ can be applied in a successful computation only when the first counter holds value zero. So an application of that rule always means redrawing the zero-mark, but never adding further lines to the picture drawn in the derivation. By a similar reason a rule of the form $q\to q'z$ can be applied only when the second cursor is at vertex zero and no new lines are added to the picture. The remaining rules use penup-mode only, so the derivation draws the picture p. This shows that (i) implies (ii). Trivially, (ii) implies (iii), so it remains to show

that (iii) implies (i).

Let an arbitrary derivation in G of a subpicture of p be given. By construction of G the first step and only the first step is an application of the initiatory rule and the last and only the last step is an application of a final rule . Omitting these two steps we can construct a computation of M according to the above table. We have to show that this is a successful computation on input (m,0) .

By definition of the initiatory rule two lines are drawn in the first step of the derivation, that is, both lines of p . In the same way as above a correspondence between the values of the counters and the positions of the cursors relative to the zero-mark is established for all steps of the computation. An application of a rule of the form $q \to q'z$ is allowed only if the second cursor is positioned exactly at the zero-mark, because in any other case a third line would be added to the picture. This position of the cursor indicates that the second counter contains the value zero, so in this situation a triple $(q,if2,q') \in \delta$ may be used in a computation of M . By an analogous argument, an application of a triple $(q,if1,q') \in \delta$ occurs only when the first counter contains value zero. The last step of the derivation is a rule of the form $q \to \lambda$, which means that the computation ends in a final state. @

From Proposition (3.1) we know that there is a two-counter-machine M such that emptiness of $f_M(m,0)$ is undecidable. Hence Theorem (3.2) implies the following

3.3 Corollary There is a linear chain code picture language for which the membership problem and the superpicture problem are undecidable.

To transfer the above results to strictly one-dimensional languages one may redefine the grammar G using rrr instead of r , lll instead of l , the zeromark $z := \downarrow rrll$, the initiatory rule $S \to l\uparrow llq_0z$ and prove the following

3.4 Theorem For every tcm M there is a linear grammar G over $\{r,l,\uparrow,\downarrow\}$ such that for every $m \in N$ the following assertions are equivalent:

(i) $f_M(m,0) \neq \emptyset$
(ii) $pic(rr\uparrow r^{3m}\downarrow r) \in Pic(G)$
(iii) $pic(rr\uparrow r^{3m}\downarrow r) \in Super(G)$

While the membership problem is decidable for context-free picture languages without invisible lines (over {u,d,r,l} [4, Theorem 5.1]) and even decidable in polynomial time for picture languages over {r,l,d} [11], we have shown that there is a linear picture language over $\{r,l,\uparrow,\downarrow\}$ for which the membership problem is undecidable. In contrast to this, [4, Theorem 5.3] can easily be transfered from languages over {u,d,r,l} to arbitrary context-free picture languages [10]. So we have the following

3.5 Fact The subpicture problem is decidable for context-free chain code picture languages.

4. DISCUSSION

It is easy to extend our concept of a picture language to more than two dimensions, to add features like colours or diagonal lines, or to switch to a triangle grid instead of the square grid: All such changes have no effect on the presented results. It is more interesting to consider the restricted class of "stripe" picture languages, which are essentially one-dimensional. Regular picture languages consisting of pictures all fitting into a stripe of fixed width are "easy", since they can be simulated by regular string languages [8, Lemma 5.8], so membership problem, subpicture problem, and superpicture problem are decidable for these languages in linear time [9]. But for a strictly one-dimensional picture language generated by a linear grammar we have shown undecidability of the membership and the superpicture problem. Are there any language classes which are significantly "richer" than regular

languages such that the membership problem is decidable? For the class of languages accepted by automata with a finite control and one counter it is straight forward to prove undecidability in the same manner as in Theorem 3.2 (draw the picture using one cursor, its position represents a second counter).

The length of a shortest description of a picture in a regular language is quadraticaly bounded in the extension of the described picture [9]. A similar result for linear and for context-free languages without penup is known [8], but for linear and for context-free picture languages in general no such bound can exist, since any such bound would immediately give rise to a decision procedure for the membership problem.

REFERENCES

[1] Rosenfeld, A.: Picture Languages - Formal Models of Picture Recognition, Academic Press, London (1979).

[2] Freeman, H.: Computer Processing of Line-Drawing Images, Comput. Surveys 6 (1974), 57-97.

[3] Kim, C. and Sudborough, I.H.: The Membership and Equivalence Problems for Picture Languages, TCS 52 (1987), 177-192.

[4] Maurer, H.A., Rozenberg, G., Welzl, E.: Using String Languages to Describe Picture Languages, Inform. and Contr. 54 (1982), 155-185.

[5] Minski, M.L.: Recursive Unsolvability of Post's Problem of "Tag" and other topics in Theory of Turing Machines, Annals of Mathematics 74 (1961), 437-455.

[6] Mylopoulus, J.: On the Application of Formal Language and Automata Theory to Pattern Recognition, Pattern Recognition 4 (1972), 37-51.

[7] Shaw, A.C.: A Formal Picture Description Scheme as a Basis for Picture Processing Systems, Inform. and Contr. 14 (1969), 9-52.

[8] Sudborough, I.H. and Welzl, E: Complexity and Decidability for Chain Code Picture Languages, TCS 36 (1985), 175-202.

[9] Hinz, F. and Welzl, E.: Regular Chain Code Picture Languages with Invisible Lines, Graz University of Technology, Institute fuer Informationsverarbeitung, Report 252 (1988).

[10] Hinz, F: Kontextfreie Kettenbildsprachen, in preparation.

[11] Kim, C.: Complexity and Decidablility for Restricted Classes of Picture Languages, manuscript.

IMYCS'88, Smolenice Castle, November 14-18, 1988.

ON AUTOMATA WITH VARIABLE NUMBER OF HEADS

Juraj Hromkovič, Ladislav Janiga, Václav Koubek

Dept. of Theor. Cybernetics, Comenius University,
842 15 Bratislava, Czechoslovakia

Moravanu 68, 169 00 Praha 6, Czechoslovakia

Computing Centre, Charles University,
Malostranské nám. 25, 118 00 Praha 1, Czechoslovakia

1. ABSTRACT

We investigate multihead automata whose number of heads is given as a function of size of input data to be analyzed. We relate the computational power of these automata with the power of other types of machines. We extend some results known for multihead automata with fixed number of heads to the case of variable number of heads. In this draft we give only outlines of the proofs.

2. INTRODUCTION

Multihead automata represents a simple and commonly investigated model of computing machines. Even if they are rather simple and in general less powerfull than other models of computing machines they are rather difficult to analyze. There are several problems related with classes of languages recognized by multihead automata, e.g. the problem whether the deterministic automata recognize the same languages as the nondeterministic ones (version of LBA). The

automata are a good model of a machine operating over a rather large data when it can access only a limited portion of them at each moment of computation. Automata with fixed number of heads are of concern when the size of the limited accessible portion of data is fixed too, in the other case when the size of accessible portion can vary as a function of size of input data the automata with variable number of heads become to be more interesting. In this paper we define them formally and we investigate their properties. We relate them to space bounded Turing machines and extend several results known for machines with fixed number of heads. Recall for completeness that the multiheada automata with fixed number of heads correspond fully to logspace bounded Turing machines (can simulate each other preserving determinancy if necessary), see |1|. For precise definitions and commonly known facts see |1,2|.

3. DEFINITIONS AND BASIC FACTS

In this section we define automata with variable number of heads. The formal definition is obvious, the resulting problem is that the computational power of such automata can be represented not only by the number of heads they are allowed to use but can be hidden in the "next step" function as well. This is clear since the next step function must manage a possibly growing number of heads and cannot be represented by a finite table. During this paper we are interested in deterministic automata but the cases of nondeterministic ones are fully analogous, the differences are pointed out if necessary. The following definition introduces automata with variable number of heads.

Definition 1. f(n)-headed automata and Turing machines.

(1) Let f be a nondecreasing function from N to N. An **f(n)-headed automaton M** is the device which has an input tape containing the input word between two endmarkers ("#" and "$") and on which f(n) heads move independently (provided the length of actual input word is n) under the following control:

(a) M has a finite state space Q with a starting state q_0 and the set of final states F distinguished.

(b) M has a transition function δ which for every input of length n is a function of the form:

$$A^{f(n)} \times Q \to Q \times \{left, right, none\}^{f(n)}.$$

(c) there exists a Mealy automaton such that input and output heads can do ε-moves computing δ.

Every steps of M's computation over an input word is characterized by the M's current state and the ordered collection of symbols read by its heads. M being in state q and reading $a_1,\ldots,a_{f(n)}$ by its heads operates so that if $\delta(a_1,\ldots,a_{f(n)},q) = \langle p, move_1,\ldots,move_{f(n)}\rangle$ then the new M's state will be p and all heads move according to the corresponding move value (moves outside the area contained between endmarkers are not allowed). M starts in its starting state having all heads on the left endmarker and accepts when reaches by all of its heads the right endmarker in a final state from F.

(2) The language L(M) **recognized by M** is the set of all words w from A^* such that M accepts \$w#.

(3) We say that a head is **one-way** if its corresponding move value can be "right" or "none" only.

(4) A head is said to be **blind** if it is able to recognize endmarkers only (precisely it means that the transition function does not depend on values read by the head except for "#" and "\$").

(5) For **real-time automata** we reqest only reaching of a final state for accepting and input word.

(6) Analogously we define **f(n)-headed h(n)-space bounded Turing machines**, such machines have one input tape on which a two-way read-only head moves and a working tape on which move f(n) independent heads (provided input tape contains a word of length n) within an area of length h(n) contained between endmarkers "\$" and "#" (thus \$ moves on the working tape). The heads are allowed to change the content of the cell they are scanning as well.

(7) We say that the **transition function** of a f(n)-headed Turing machine is of **complexity** h(n) if there is a Turing machine computing it in space bound h(n) where n is the size of input.

(8) **Nondeterministic machines (or automata)** are defined in a standard way using transition relation instead of function. In this case automata can make a choise how to continue.

(9) For all automata we suppose that **conflicting situations** (two heads want to rewrite a unique cell in a different manner,...) result in halting and rejecting the input.

(10) For a function f, **f-DFA** will denote the set of all f(n)-headed finite state automata. **(f,g)-TM** will stand for the class of f(n)-headed g(n) space bounded Turing machines. Further, **(f,g,h)-TM** denotes f(n)-headed, g(n) space bounded Turing machine with the transition function of complexity h(n). The corresponding classes of nondeterministic machines are denoted by **f-NFA, (f,g)-NTM,** and **(f,g,h)-NTM.**

The followig two theorems are trivial consequences of the fact that the transition function can hide a great deal of computational power. Denote by id the identity function of the set of all natural numbers, denote by $\underset{\sim}{n}$ the constant mapping of the set of all natural numbers into itself to n.

Theorem 2. For every (even nonrecursive) language L there is a $(id,\underset{\sim}{0})$-TM recognizing L in real time.

Theorem 3. For every unbounded function f there is a language which is not recurively enumerable and which is recognized by some $(f,\underset{\sim}{0})$-DFA in real time.

Theorem 4. For every language $L \in SPACE(f(n))$ there exists a function $g(n) = 0(f(n)/\log n)$ and there exists a g-DFA with one one-way head and all other heads blind such that L(M)=L.

Theorem 5. Let M be a g-DFA, then $L(M) \in SPACE(g(n)\log n)$.

Theorem 6. Let M be a (f,g,h)-TM then
$L(M) \in SPACE\{f(n)\log(g(n)),g(n),h(n)\}$.

Theorem 7. Let $L \in SPACE(f(n))$, and g,h be nondecreasing functions with $g(n)/\log(h(n)) = \Omega(f(n))$. Then there exists a $(g,h,\underset{\sim}{1})$-TM with blind heads on the working tape and one one-way input head such that L = L(M).

The crutial point when investigating automata with increasing number of heads consists in the fact that the complexity of the transition functi-

ons hides a great deal of computational power of these automata. To the end of this section we summarize under which (restrictive) conditions our results will hold as well. First one can derive that it is immaterial whether the heads can recognize an eventual coincidence on the input tape. Moreover, it is immaterial whether the set of symbols read by corresponding heads is supposed to be ordered (each symbol being labelled by the head by which it is scanned) or not (only information about what symbols are currently read but not by which heads). In this case the information about what heads have been moving in the preceding step is required, too.

One more substantial restriction consists in the fact that one can manage a growing number of heads in a very similar manner to the case when the number is fixed beforehand. It can be done when only a limited number of heads are managed freely and the rest of the heads are managed in common, i.e. by defining what is the common behaviour of all "other" heads. The more precise definition of such "symetrical or quasi finite" automata follows.

Definition 8. Symetrical and quasi-finite f-head automata.
An f(n)-headed automaton M is said to be **symetrical** if the transition function δ is a mapping from $A^{f(n)} \times Q \times \{left, right, none\}^{f(n)}$ such that for every $\alpha \in Q \times \{left,right,none\}^{f(n)}$ for every $a_1,a_2,\ldots,a_{f(n)} \in A$, and every permutation π of the set $\{1,2,\ldots,f(n)\}$ we have $\delta(a_1,\ldots,a_{f(n)},\alpha) = (a_{\pi(1)},\ldots, a_{\pi(f(n))},\alpha)$. If the current state of M is q, the i-th head reads the letter a_i and the previous move of the i-th head was $move_i$ then the next configuration of M is determined by $\delta(a_1,\ldots,a_{f(n)},q,move_i,\ldots,move_{f(n)})$. At the beginning we assume the previous move of every head was "none".

An f(n)-headed automaton M is said to be **k-quasi-finite** f(n)-headed where k is a natural number iff the transition function of M fulfils the following:

(1) At every step of computation of M on an input x of length n there are exactly k heads (active heads) which are managed by a finite state control. The remaining (f(n)-k) heads are managed in common so that they are all allowed in the same time to continue in previous moves,

stop, move to the right or to the left, the symbols they are scanning being completely immaterial for the function of M.

(2) The heads of M are numbered from 1 through f(n) and the set of k active heads is updated dynamically at every step so that every head i which is active can be replaced by some inactive head with greatest number less than i or with the smallest number greater than i depending on the choice.

(3) The starting set of active heads of M is $\{1,\ldots,k\}$.

Corollary 9. For every $L \in SPACE(f(n))$ there exist $g = O(f(n)/\log n)$ and a natural number k such that there exist a symmetrical g-DFA M_1 and a k-quasi-finite g-DFA M_2 with one one-way head and all other heads blind such that $L(M_1) = L(M_2) = L$.

4. REAL-TIME f(n)-HEADED AUTOMATA

In this section we generalize results for automata with fixed number of heads from |3| and obtain full analogies for f(n)-headed automata for f sufficiently small functions (growing slowly).

Definition 10. Let f be a function. Then R(f) will denote the class of all languages recognized by deterministic f(n)-headed automata in real-time. NR(f) will stand for the nondeterministic analogy of R(f). If g is another (nondecreasing) function then NR(f,g) will stand for the classof all languages recognized by f(n)-headed automata using at most g(n) nondeterministic steps (provided the size of input is n).

For two functions f and g, $f<g$ will denote the fact that f grows slowly than g (i.e. $\lim f(n)/g(n) = 0$).

Theorem 11. Let $f<g\leq\log$. Then R(f) respectively NR(f) is properly contained in R(g) respectively NR(g).
Let $g<h\leq\log$ and $f>g$. Then for $f<\log$ we have NR(f,g) is properly contained in NR(f,h).

P r o o f . These results follows from analogies from |3| where they are proved for automata with fixed number of heads. While

the proofs in |3| are based on languages containing series of fixed length of palindromes separated by some markers the above results are obtained when series of increasing length are considered. The basic results from |3,4| state that an automaton which has to recognize a language containing series of palindromes of length k must have at least $k+1$ heads and the proofs work for the case of unlimited number of heads as well. In this case the fact that an automaton can recognize series of $h(n)$ palindromes only if the number of its heads is not essentially smaller than $h(n)$.follows.

5. LOWER BOUND TECHNIQUE FOR DEVICES WITH WARIABLE NUMBER OF HEADS.

The last section of this paper is devoted to the development of a lower bound technique for devices with variable number of heads. The lower bound technique introduced belowis a generalization of crossing sequence technique presented for multihead Turing machines in |5,8| and multihead automata in |3,6,7,9|. We note that it is independent on the complexity of transition function.

Let, for any computing device A, H_A, S_A, T_A be functions denoting the head, space, and time complexity of A respectively.

Let $\oplus$ be the Boolean sum operator (sum mod 2). Let

$$S = \{x_1 2^m x_2 2^m \ldots 2^m x_r 2^z \mid x_i \in \{0,1\}^m, \ \sum_{i=1}^{r}{}^{\oplus} x_i = 0^m, \ m \geq 1, \ r \geq 1, \ z \geq 1\},$$

where $\sum_{i=1}^{r}{}^{\oplus} x_i = 0^m$ means that $x_{1j} \oplus \ldots \oplus x_{rj} = 0$ for $j = 1,\ldots,m$ and $x_i = x_{i1}\ldots x_{im}$ for $i = 1,\ldots,r$.

Let f and g be some functions from N to N such that $n-2f(n)g(n) \geq 0$ for all $n \in N$. We shall consider languages

$$S(f,g) = \{x_1 2^{g(n)} x_2 2^{g(n)} \ldots 2^{g(n)} x_{f(n)} 2^{g(n)+z(n)} \mid n \geq 1, \ x_i \in \{0,1\}^{g(n)} \text{ for } i = 1,\ldots,f(n), \ z(n) = n - 2f(n)g(n), \ \sum_{i=1}^{f(n)}{}^{\oplus} x_i = 0^{g(n)}\} \subseteq S.$$

Theorem 12. Let A be a nondeterministic Turing machine such that $S(f,g) \subseteq L(A) \subseteq S$ for some functions f, and g from N to N

such that $n/2 \geq n-2f(n)g(n) \geq 0$. Let A has q states and k working symbols. Then there is an $m \in N$ such that for all $n \geq m$

(i) $32\log_2(\max\{k,q\})H_A(n)T_A(n)(H_A(n)\log_2 n+S_A(n)) \geq ng(n)$ or

(ii) $32(H_A(n))^3 T_A(n) \geq nf(n)$.

Corollary 13. Let A be a nondeterministic Turing macine such that $L(A) = S(\lfloor n^{1/2} \rfloor, \lfloor n^{1/2}/2 \rfloor)$. Then

$$H_A(n)T_A(n)((H_A(n))^2 + H_A(n)\log_2 n + S_A(n)) \in \Omega(n^{3/2}).$$

Concluding this paper let us show that the lower bound presented above is almost optimal.

Lemma 14. There is an f(n)-headed automaton recognizing S(f,g) in linear time for any function g constructible in linear time by f(n)-headed automata.

Lemma 15. There is a one-headed g(n)-space bounded Turing machine recognizing S(f,g) in linear time for any linear time, g(n) space constructible function f .

REFERENCES.

1. Hartmanis,J.: On nondeterminancy in simple computing devices. Acta Informatica 1 (1972).
2. Hopcroft,J.E. - Ulmann,J.D.: Formal languages and their relations to automata. Addison-Wesley 1968.
3. Janiga,L.: An other hierarchy defined by multihead finite automata. In: Proc. MFCS´81, Lecture Notes in Computer Science 118, Springer-Verlag, Berlin-Heidelberg-New York 1981.
4. Janiga,L.: Real-time computations of two-way multihead finite automata. In: Proc. FCT´79, Academie-Verlag, Berlin 1979.
5. Ďuriš,P. - Galil,Z.: A time-space tradeoff for language recognition. Math. Syst. Theory 17 (1984), 3-12.
6. Hromkovič,J.: Fooling a two-way nondeterministic multihead automaton with reversal number restriction. Acta Informatica 22 (1985), 589-594.
7. Hromkovič,J.: On the power of alternation in automata theory. J. of Computer and System Sciences 31 (1985), No.1, 28-39.
8. Hromkovič,J.: Tradeoffs for language recognition on parallel computing models. In: Proc. 13th ICALP´86, Lecture Notes in Computer Science 226, Springer-Verlag, Berlin-Heidelberg-New York 1986, pp. 157-166.
9. Rivest,R.L. - Yao,A.C.: k+1 heads are better than k. J. of ACM 25 (1978), No.2, 337-340.

IMYCS'88, Smolenice Castle, November 14-18, 1988.

The Simulation of Two-Dimensional One-Marker Automata by Three-Way Two-Dimensional Turing Machines

Akira ITO+
Katsushi INOUE++
and
Itsuo TAKANAMI++

+ *Technical College*
++*Faculty of Engineering*
Yamaguchi University
Ube, 755 Japan

Introduction & Preliminaries We denote a two-dimensional deterministic (nondeterministic) one-marker automaton by "2-DM_1" ("2-NM_1"), and a three-way two-dimensional deterministic (nondeterministic) Turing machine by "TR2-DTM" ("TR2-NTM"). In this paper, we investigate the necessary and sufficient space for TR2-NTM's and TR2-DTM's to simulate 2-DM_1's and 2-NM_1's. The results are shown in Table 1, where n is the number of columns of rectangular input tapes. Compare those with the case of two-dimensional deterministic (nondeterministic) finite automaton which is denoted by "2-DF" ("2-NF") in the table.

Table 1. Necessary and sufficient space for Ys to simulate Xs.

(a) X is 0-marker [3-5].

X\Y	TR2-DTM	TR2-NTM
2-DF	$\Theta(n \log n)$	$\Theta(n)$
2-NF	$\Theta(n^2)$	$\Theta(n)$

(b) X is 1-marker.

X\Y	TR2-DTM	TR2-NTM
2-DM_1	$2^{\Theta(n \log n)}$	$\Theta(n \log n)$
2-NM_1	$2^{\Theta(n^2)}$	$\Theta(n^2)$

In this paper, the detailed definitions of two-dimensional marker automata and (space-bounded) three-way two-dimensional Turing machines are omitted. If necessary, refer to [1,5].

Definition 2.1. Let Σ be a finite set of symbols. A *two-dimensional tape* over Σ is a two-dimensional rectangular array of elements of Σ.

The set of all two-dimensional tapes over Σ is denoted by $\Sigma^{(2)}$.

For a tape $x \in \Sigma^{(2)}$, we let $\ell_1(x)$ be the number of rows of x and $\ell_2(x)$ be the number of columns of x. If $1 \leq i \leq \ell_1(x)$ and $1 \leq j \leq \ell_2(x)$, we let $x(i,j)$ denote the symbol in x with coordinates (i,j). Furthermore, we define

$$x[(i,j),(i',j')],$$

when $1 \leq i \leq i' \leq \ell_1(x)$ and $1 \leq j \leq j' \leq \ell_2(x)$, as the two-dimensional tape z satisfying the following:

(i) $\ell_1(z)=i'-i+1$ and $\ell_2(z)=j'-j+1$,

(ii) for each k,r $[1 \leq k \leq \ell_1(z), 1 \leq r \leq \ell_2(z)]$, $z(k,r)=x(k+i-1,r+j-1)$.

When a two-dimensional tape x is given to any two-dimensional automaton as an input, x is surrounded by the boundary symbol "#"s.

Definition 2.2. Let x be in $\Sigma^{(2)}$ and $\ell_2(x)=n$. When $\ell_1(x)$ is divided by n, we call

$$x[((j-1)n+1,1),(jn,n)]$$

an *n-block of x*, for each $j(1 \leq j \leq \ell_1(x)/n)$.

Definition 2.3. For any two-dimensional automaton M with input alphabet Σ, define $T(M)=\{x \in \Sigma^{(2)} \mid M \text{ accepts } x\}$. Furthermore, define

$\mathcal{L}[2\text{-DM}_1]=\{T \mid T=T(M) \text{ for some 2-DM}_1 \text{ M}\}$ and

$\mathcal{L}[2\text{-NM}_1]=\{T \mid T=T(M) \text{ for some 2-NM}_1 \text{ M}\}$.

We similarly define $\mathcal{L}[\text{TR2-DTM(L(m,n))}]$ ($\mathcal{L}[\text{TR2-NTM(L(m,n))}]$) as the class of sets accepted by L(m,n) space-bounded TR2-DTMs (TR2-NTMs).

By using an ordinary technique, We can easily show that the following theorem holds.

Theorem 2.1. *For any function $L(n) \geq \log n$,*

$$\mathcal{L}[\text{TR2-NTM(L(n))}] \subseteq \mathcal{L}[\text{TR2-DTM}(2^{O(L(n))})].$$

3. Sufficient Space.

We first show that n log n space is sufficient for TR2-NTM's to simulate 2-DM_1's.

Theorem 3.1. *$\mathcal{L}$[2-DM_1] $\subseteq$ $\mathcal{L}$[TR2-NTM(n log n)].*

Proof. Suppose that a 2-DM_1 M is given. Let the set of states of M be S. We partition S into two disjoint subsets S^+ and S^- which corresponds to the sets of states when M is holding and not holding the marker in the finite control, respectively. (Rigorously, neither S^+ nor S^- contains the states in which the input head of M positions on the same cell as where the marker is placed.) We assume that the initial state q_0 and the unique accepting state q_a of M are both in S^+. In order to make our proof clear, we also assume that M begins to move with its input head on the rightmost bottom boundary symbol # of an input tape and, when M accepts an input, it enters the accepting state at the rightmost bottom boundary symbol.

Suppose that an input tape x with $\mathcal{Q}_1(x)=m$ and $\mathcal{Q}_2(x)=n$ is given to M. For M and x, we define three types of mappings $f^{\dagger -}_i: S^- \times \{0,1,\ldots,n+1\} \to S^- \times \{0,1,\ldots,n+1\} \cup \{\mathcal{Q}\}$, $f^{\dagger +}_i: S^+ \times \{0,1,\ldots n+1\} \to S^+ \times \{0,1,\ldots,n+1\} \cup \{\mathcal{Q}\}$, and $f^{\downarrow -}_i: S^- \times \{0,1,\ldots,n+1\} \to S^- \times \{0,1,\ldots,n+1\} \cup \{\mathcal{Q}\}$ (i=0,1,...,m+1) as follows.

$$f^{\dagger -}_i(q^-,j) = \begin{cases} (q^{-\prime},j'): & \text{Suppose that we make M start from the configuration } (q^-,(i-1,j)) \text{ with no marker on the input x (i.e., we take away the marker from the input tape by force). After that, if M reaches the i-th row of x in some time, the configuration corresponding to the first arrival is } (q^{-\prime},(i,j')); \\ \mathcal{Q}: & \text{Starting from the configuration } (q^-,(i-1,j)) \text{ with no marker on the input tape, M never reaches the i-th row of x.} \end{cases}$$

$$f^{\dagger +}_i(q^+,j) = \begin{cases} (q^{+\prime},j'): & \text{Suppose that we make M start from the configuration } (q^+,(i-1,j)). \text{ After that, if M reaches the i-th row of x with its marker held in the finite control in some time (so, when M puts down the marker on the way, it} \end{cases}$$

must return to this position again and pick up the marker), the configuration corresponding to the first arrival is $(q^+',(i,j'))$;

Λ : Starting from the configuration $(q^+,(i-1,j))$ with no marker on the tape, M never reaches the i-th row of x with its marker held in the finite control.

$f^{\downarrow -}{}_i(q^-,j)=$ (q^-',j'): Suppose that we make M start from the configuration $(q^-,(i+1,j))$ with no marker on the input tape (i.e., we take away the marker from the input tape by force). After that, if M reaches the i-th row of x in some time, the configuration corresponding to the first arrival is $(q^-',(i,j'))$,

Λ : Starting from the configuration $(q^-,(i+1,j))$ with no marker on the tape, M never reaches the i-th row of x.

Below, we show that there exists a TR2-NTM(n log n) M' such that T(M')=T(M). Roughly speaking, while scanning from the top row down to the bottom row of the input, M' guesses and checks $f^{\downarrow -}{}_i$, constructs $f^{\uparrow -}{}_i$ and $f^{\uparrow +}{}_i$, and finally at the bottom row of the input, M' decides by using $f^{\uparrow -}{}_{m+1}$ and $f^{\uparrow +}{}_{m+1}$ whether or not M accepts x. (See Figure 1.) In order to record these mappings for each i, O(n) blocks of O(log n) size suffice, so totally O(n log n) cells of the working tape suffice. More precisely, the working tape must be used as a "multi-track" tape. In the following discussion, we omit the detailed construction of the working tape of M'.

First, set $f^{\uparrow -}{}_0$, $f^{\uparrow +}{}_0$ to the fixed value Λ.

For i=0 to m+1, repeat the following. [$f^{\uparrow -}{}_i$, $f^{\uparrow +}{}_i$ are already computed at the (i-1)st row.]

(0) Go to the i-th row; When i=0, assume the boundary symbols on the first row.

(1) Guess $f^{\downarrow -}{}_i$; if i=m+1, set $f^{\downarrow -}{}_{m+1}$ to the fixed value Λ.

(2) [compute $f^{\uparrow -}{}_{i+1}$ from $f^{\uparrow -}{}_i$] When $i\neq m+1$, do the following: Assume

that there is no marker on the input tape. For each $(q^-,j)\in S^-\times\{0,1,\ldots,n+1\}$, start to simulate M from the configuration $(q^-,(i,j))$. While M moves only at the i-th row, behave just as M does. On the way of the simulation, if M would go up to the (i-1)st row at the k-th column and would enter the internal state p^-, then search the table $f^{\uparrow -}{}_i$ to know the behavior of M above the i-th row. If the value $f^{\uparrow -}{}_i(p^-,k)$ is "Ω", write "Ω" into the block corresponding to $f^{\uparrow -}{}_{i+1}(q^-,j)$; If the value $f^{\uparrow -}{}_i(p^-,k)$ is "(p^-,k')", restart the simulation of M from the configuration $(p^-,(i,k'))$. While continuing to move in this way, if M would go down to the (i+1)st row, then write the pair of the internal state and column number just after that movement into the block corresponding to $f^{\uparrow -}{}_{i+1}(q^-,j)$ of the working tape. If M never goes down to the (i+1)st row (including the case when M enters a loop), then write "Ω" into the correspondent block.

(3) [compute $f^{\uparrow +}{}_{i+1}$ from $f^{\uparrow -}{}_i$, $f^{\uparrow +}{}_i$, and $f^{\downarrow -}{}_i$] When $i\neq m+1$, do the following: For each $(q^+,j)\in S^+\times\{0,1,\ldots,n+1\}$, starting from the configuration $(q^+,(i,j))$, simulate M until M goes down to the (i+1)st row with the marker in the finite control. On the way of the simulation, if M would go up to the (i-1)st row with the marker held, then search the table $f^{\uparrow +}{}_i$ to know the behavior of M above the i-th row. If this value of $f^{\uparrow +}{}_i$ is "Ω", write "Ω" into the block corresponding to $f^{\uparrow +}{}_{i+1}(q^-,j)$; otherwise, restart the simulation of M from the configuration on the i-th row determined by the table value. If M puts the marker down on the i-th row of the input tape, then record the column number of this position in some track of the working tape and start the simulation of M which has no marker in the finite control. After that, If M would go down to the (i+1)st row or would go up to the (i-1)st row, then search the respective table $f^{\downarrow -}{}_i$ or $f^{\uparrow -}{}_i$ to find the configuration in which M return to the i-th row again. (If M never returns to the i-th row, write "Ω" into the block corresponding to $f^{\uparrow +}{}_{i+1}(q^+,j)$). From this configuration, restart the simulation of M. After that, if M returns to the position where M put down the marker previously and picks it up, then continue the simulation of M; otherwise write "Ω" into the block corresponding to (q^+,j). At some point of the simulation, If M goes down to the (i+1)st row with the marker held

in the finite control, write the pair of the internal state which M would enter just after that time and the row number of this head position into the block corresponding to $f^{\uparrow +}{}_{i+1}(q^+,j)$. If M never goes down to the (i+1)st row, then write "Ω" into the correspondent block.

(4) [check $f^{\downarrow -}{}_{i-1}$ from $f^{\downarrow -}{}_i$] When $i\neq 0$, do the following: In order to check that the table $f^{\downarrow -}{}_{i-1}$ guessed on the previous row is consistent with the table $f^{\downarrow -}{}_i$ (guessed at the present row), first newly compute a mapping $\underline{f^{\downarrow -}}{}_{i-1}$, which is uniquely determined from $f^{\downarrow -}{}_i$ and the content of the i-th row of the input. [Assume that there is no marker on the input tape. For each $(q^-,j)\in S^-\times\{0,1,\ldots,n+1\}$, M' starts to simulate M from the configuration $(q^-,(i,j))$. While M moves only at the i-th row, M' behaves just as M does. On the way of the simulation, if M would go down to the (i+1)st row at the k-th column and would enter the internal state p^-, then M' searches the table $f^{\downarrow -}{}_i$ to know the behavior of M below the i-th row. If the value $f^{\downarrow -}{}_i(p^-,k)$ is "Ω", M' writes "Ω" into the block corresponding to $\underline{f^{\downarrow -}}{}_{i-1}(q^-,j)$; If the value $f^{\downarrow -}{}_i(p^-,k)$ is "$(p^{-\prime},k')$", M' restarts the simulation of M from the configuration $(p^{-\prime},(i,k'))$. While continuing to move in this way, if M would go up to the (i-1)st row, then M' writes the pair of the internal state and column number just after that movement into the block corresponding to $\underline{f^{\downarrow -}}{}_{i-1}(q^-,j)$ of the working tape. If M never goes up to the (i-1)st row (including the case when M enters a loop), then M' writes "Ω" into the correspondent block.] After this computation, check that $\underline{f^{\downarrow -}}{}_{i-1}$ is identical to the mapping $f^{\downarrow -}{}_{i-1}$ guessed at the previous row. If the equality holds, then continue the process; otherwise, reject and halt.

After the above procedure, on the (m+1)st row, M' begins to simulate M from the initial configuration $(q^+{}_0,(m+1,n+1))$ to decide whether or not M accepts the input after all. When M goes up to the m-th row with or without the marker, we can know how M returns again to the (m+1)st row, from $f^{\uparrow +}{}_{m+1}$ or $f^{\uparrow +}{}_{m+1}$, respectively. If M never returns to the (m+1)st row again, then M' rejects and halts. If M returns to the (m+1)th row, then M' continues the simulation. M' accepts the input x only if M' finds that M enters the accepting configuration $(q^+{}_a,(m+1,n+1))$.

It will be obvious that T(M)=T(M').

From Theorem 2.1 and Theorem 3.1, we get the following.

Corollary 3.1. $\mathcal{L}[2\text{-}DM_1] \subseteq \mathcal{L}[TR2\text{-}DTM(2^{O(n \log n)})]$.

We next show that n^2 space is sufficient for TR2-NTM's to simulate 2-NM_1's. The basic idea and outline of the proof are the same as those of Theorem 3.1.

Theorem 3.2. $\mathcal{L}[2\text{-}NM_1] \subseteq \mathcal{L}[TR2\text{-}NTM(n^2)]$.

Proof. Suppose that a 2-NM_1 M and an input x with $Q_1(x)=m$ and $Q_2(x)=n$ are given. We take the same assumptions and notations for the states of M, initial and accepting configurations of M as in the proof of Theorem 3.1.

From M and x, we define three types of mappings $g^{\dagger -}{}_i : S^- \times \{0,1,\ldots,n+1\} \to 2^{S^- \times \{0,1,\ldots,n+1\}}$, $g^{\dagger +}{}_i : S^+ \times \{0,1,\ldots n+1\} \to 2^{S^+ \times \{0,1,\ldots,n+1\}}$, and $g^{\downarrow -}{}_i : S^- \times \{0,1,\ldots,n+1\} \to 2^{S^- \times \{0,1,\ldots,n+1\}}$ (i=0,1,...,m+1) as follows.

$g^{\dagger -}{}_i(q^-,j) \ni (q^{-\prime},j')$: Suppose that we take away the marker of M from the input tape. Then, there exists a sequence of moves in which M starts from the configuration $(q^-,(i-1,j))$ and reaches the i-th row of x in the configuration $(q^{-\prime},(i,j'))$ for the first time.

$g^{\dagger +}{}_i(q^+,j) \ni (q^{+\prime},j')$: There exists a sequence of moves in which M starts from the configuration $(q^+,(i-1,j))$ and reaches the i-th row of x in the configuration $(q^{+\prime},(i,j'))$ for the first time with its marker in the finite control (so, when M puts down the marker on the way, there exists a sequence of moves in which M returns to this position and picks up the marker).

$g^{\downarrow -}{}_i(q^-,j) \ni (q^{-\prime},j')$: Suppose that we take away the marker from the input tape. Then, there exists a sequence of moves in which M starts from the configuration $(q^-,(i+1,j))$ and M reaches the i-th row of x in the configuration $(q^{-\prime},(i,j'))$ for the first time.

Note that, in order to record these mappings for each i, totally $O(n^2)$

cells of working tape suffice. Roughly speaking, a TR2-NTM M' accepting T(M) acts as follows: While scanning from the top row down to the bottom row of the input, M' guesses and checks $g^{\downarrow -}{}_i$, constructs $g^{\uparrow -}{}_i$ and $g^{\uparrow +}{}_i$, and finally at the bottom row of the input, M' decides whether or not M accepts x by using $g^{\uparrow -}{}_{m+1}$ and $g^{\uparrow +}{}_{m+1}$. The precise constructions of M' are ommited here.

■

From Theorem 2.1 and Theorem 3.2, we get the following.

Corollary 3.2. $\mathcal{L}[2\text{-}NM_1] \subseteq \mathcal{L}[TR2\text{-}DTM(2^{O(n^2)})]$.

4. Necessary space.

In this section, we show that the algorithms described in the previous section are optimal in some sense. That is, those spaces are required for three-way Turing machines when the spaces depend only on one variable n (i.e., the number of columns of the input tapes).

Lemma 4.1. Let $T_1=\{x\in\{0,1\}^{(2)} \mid \exists n\geq 1[\Omega_2(x)=n$ & (each row of x contains exactly one "1") & $\exists k\geq 2[$(x has k n-blocks) & (the last n-block is equal to some other n-block)]]$\}$. Then,

(1) $T_1\in\mathcal{L}[2\text{-}DM_1]$ and

(2) $T_1\notin\mathcal{L}[TR2\text{-}DTM(2^{L(n)})]$ (so, $T_1\notin\mathcal{L}[TR2\text{-}NTM(L(n))]$) for any $L:N\to R$ such that $lim_{n\to\infty}[L(n)/n \log n]=0$.

Proof. (1): We constructs a 2-DM_1 M accepting T_1 as follows. Given an input x with $\Omega_2(x)=x$, M first checks that each row of x contains exactly one "1" by horizontal sweep on each row and that x consists of k n-blocks for some $k\geq 2$ by zigzagging from top to bottom. Then, M tests whether some n-block is identical to the last n-block (i.e., the k-th n-block) from top to bottom by utilizing its own marker. (See Figure 2.): In some n-block and some row of this block, say j-th block and i-th row, M first puts the marker on the position where the input tape symbol is "1", then M starts a zigzag from the rightmost cell of the ith row of this block until it reaches the bottom boundary. M then goes back to "1" position on the i-th row of the last block. From this position, M vertically moves up until it encounters the marker previously put by itself or arrives at the top boundary. If M meets the marker again, then the i-th rows of the two

blocks are identical and M proceeds to check the (i+1)st row of the two blocks. If M does not meet the marker, M can conclude that the j-th n-block and the last n-block are different and must go to the next n-block to test the equality. M accepts x, if M finds that some n-block are identical to the last block (this fact is recognized from the fact that when M reaches the bottom boundary, this is the rightmost position there). It is clear that $T(M)=T_1$.

(2): By using an analogous technique to [2,5], we can show that Part (2) holds, but the proof is lengthy and ommitted here.

■

Lemma 4.2. Let $T_2=\{x\in\{0,1\}^{(2)} \mid \exists n\geq 1[Q_2(x)=n \ \& \ \exists k\geq 2[(x \text{ has } k \text{ n-blocks}) \ \& \ (\text{the last n-block is equal to some other n-block})]]\}$. Then,

(1) $T_2\in\mathcal{L}[2\text{-NM}_1]$,

(2) $T_2\notin\mathcal{L}[\text{TR2-DTM}(2^{L(n)})]$ (so, $T_2\notin\mathcal{L}[\text{TR2-NTM}(L(n))]$) for any $L:N\to R$ such that $lim_{n\to\infty}[L(n)/n^2]=0$.

Proof. It is shown in [3] that Part (1) holds. From the same reason as in the proof of Lemma 4.1(2), we ommit the proof of Part (2).

■

From Lemma 4.1 and Lemma 4.2, we can conclude as follows.

Theorem 4.1. *(1) To simulate 2-DM$_1$'s, TR2-NTM's require $\Omega(n \log n)$ space and TR2-DTM's require $2^{\Omega(n \log n)}$ space. (2) To simulate 2-NM$_1$'s, TR2-NTM's require $\Omega(n^2)$ space and TR2-DTM's require $2^{\Omega(n^2)}$ space.*

5. Discussion.

In this paper, we have investigated how much space is required and suffices for three-way Turing machines to simulate two-dimensional 1-marker automata on any vertically long input tapes. By a slight inprovement of the algorisms in the proofs of Theorem 3.1 and Theorem 3.2 when the number of rows are greater than the number of columns of input tapes and extended argument of the proofs of Lemma 3.4 and Lemma 3.5 in [5], we can get the following results: the nessecary and sufficient spaces for TR2-NTMs to simulate 2-DM$_1$'s and 2-NM$_1$'s are

$$n\cdot min\{\log m, \log n\} \quad \text{and} \quad n\cdot min\{m,n\} \quad (m\geq 2),$$

respectively, which are the most general expressions as the two-variable space-complexity function L(m,n). These results will appear in a sub-

sequent paper.

References

[1] A.Rosenfeld, *Picture Language*, Chapter 7 (Academic Press, NY, 1979).

[2] K.Inoue and A.Nakamura, Some Properties of Two-Dimensional Nondeterministic Finite Automata and Parallel Sequential Array Acceptors, *Trans. IECE Japan* Sec. D, pp.990-997 (1977).

[3] K.Morita, H.Umeo, and K.Sugata, Accepting Abilities of Offside-free Two-dimensional Marker Automata --- The simulation of Four-way Automata by Three-way Tape-bounded Turing Machines, *Tech. Rep. IECE Japan* AL79-2, pp.1-10 (1979).

[4] K.Inoue and A.Nakamura, Some Properties of Two-Dimensional On-Line Tessellation Acceptors, *Inform. Sci.* 13, pp.95-121 (1977).

[5] K.Inoue and I.Takanami, A Note on Deterministic Three-Way Tape-Bounded Two-Dimensional Turing Machines. *Inform. Sci.* 20, pp.41-55 (1980).

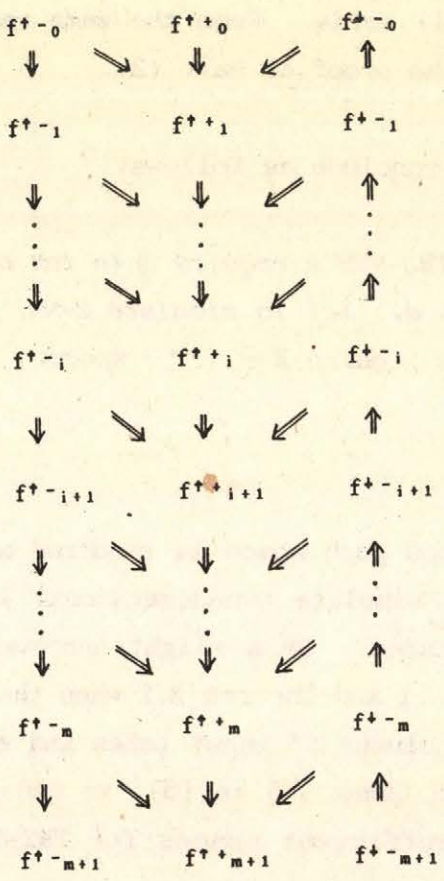


Fig.1. Mutual dependences of the mappings.

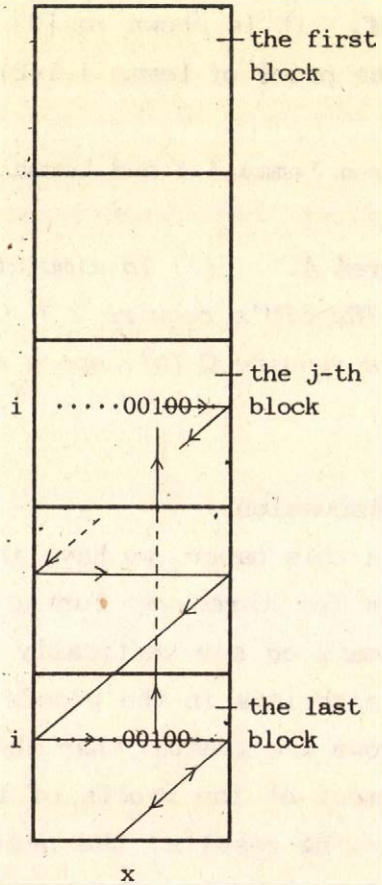


Fig.2. Action of 2-DM$_1$ M an input x in T$_1$.

IMYCS'88, Smolenice Castle, November 14-18, 1988.

SECURITY OF CIPHERING IN VIEW OF COMPLEXITY THEORY

JARKKO KARI
Mathematics Department
University of Turku
20500 Turku, Finland

1. INTRODUCTION

Since Diffie and Hellman introduced the idea of public key cryptography in (1) several new cryptosystems have been proposed. At the same time difficulties in proving the security of systems have become obvious. Problems are encountered already when trying to state what is actually meant with a secure cryptosystem. Often systems are claimed to be safe just because their breaking "seems" to require resolving some intractable problem.

This paper investigates public key cryptosystems that encrypt the plaintext in a bitwise manner using probabilistic encryption algorithms. Especially their security is discussed in view of complexity theory. We present a new public key system that is based on proposition logic. The system is shown optimal in the sense that any cryptanalytic method against it can be used to break other cryptosystems as well. Also we show that the existence of a polynomial time cryptanalytic method against the system would imply $P=NP \cap CoNP$. Finally, we discuss what these results do and what they do not mean. One purpose of the paper is to demonstrate difficulties in measuring

the safety of ciphering.

The reader is adviced to consult (3) in order to become familiar with the standard terminology of cryptography used throughout the paper.

2. PROBABILISTIC ENCRYPTION

The notion of probabilistic encryption was first introduced by Goldwasser and Micali in (2). To encrypt a message we use a fair coin. The cryptotext depends not only on the plaintext but also on the results of consecutive coin tosses. There are thus many possible cryptotexts for each plaintext. To simplify the discussion a rather restricted notion for a cryptosystem will be used.

A cryptosystem consists of the following components:

- a key space $K \subseteq K_e \times K_d$. The keys are pairs (e,d) where e is the public encryption key in K_e and d is the corresponding secret decryption key in K_d. Both keys are represented by words over some fixed alphabet. With the size of a key (e,d) we mean the sum of the lengths of e and d as words.

- a cryptotext space C.

- encryption algorithms E_0 and E_1. Given an encryption key $e \in K_e$ and a sequence $x \in \{0,1\}^*$ representing the coin tosses, they compute a cryptotext $c \in C$. The cryptotext corresponding to bit i (= 0 or 1) and key (e,d) is any $E_i(e,x)$ where x is a binary word. Because we want the encryption to be fast the algorithms E_0 and E_1 should work in polynomial time with respect to the size of the key (e,d).

- a decryption algorithm D computing a function from $K_d \times C$ to {0,1}, such that for each key (e,d), index i and sequence x of coin tosses we have $D(d,E_i(e,x))=i$. The algorithm D works in time polynimial in the size of the key (e,d).

Consider, for example, the well known quadratic residue system of Goldwasser and Micali (2). In that system the private key d consists of two

prime numbers p and q and the corresponding public key is their product N=pq. In this case the algorithms E_0, E_1 and D work indeed in polynomial time with respect to logN, the size of the key.

3. COMPLEXITY OF CRYPTANALYSIS

This section investigates the problem encountered by a cryptanalyst. Suppose the eavesdropper knows only the public encryption key e and a cryptotext c. He wants to find out what is the bit that c is obtained from. The pair (e,c) forms a legal instance for cryptanalysisproblem. There are also illegal instances, that is, either e is not a possible encryption key, or c can not be obtained with e from either bit 0 or 1. The set of such instances is denoted by ILLEGAL, while LEGAL is the set of legal instances. The set LEGAL is further divided into two subsets, BIT_0 and BIT_1. The set BIT_i consists of legal instances (e,c) where $c=E_i(e,x)$ for some x.

Recognizing instances that belong to BIT_0 or BIT_1 can be very hard - as hard as distinguishing LEGAL from ILLEGAL. But the cryptanalyst is not interested in how his method works on illegal instances because he knows his instances belong to LEGAL. For example in case of the quadratic residue system he does not need to verify that e is indeed a product of two prime numbers and not a product of three or four primes. So the cryptanalyst wishes to be able to recognize any sets L_0 and L_1 such that

$$(*) \qquad BIT_i = L_i \cap LEGAL \quad \text{for } i=0 \text{ and } 1.$$

The following theorem is an immediate consequence of our definition for a cryptosystem.

Theorem 1. For any cryptosystem there are L_0 and L_1 in NP satisfying (*).

Proof. Let L_i be the set of instances (e,c) recognized by the following nondeterministic algorithm: Guess a sequence x of coin tosses. If $E_i(e,x)=c$ then accept (e,c), otherwise reject it. Obviously, the instances (e,c) of LEGAL that are accepted by this algorithm are exactly the instances in BIT_i. What happens on instances in ILLEGAL is not important. □

4. AN NP∩CoNP-SAFE CRYPTOSYSTEM

In this section a cryptosystem based on proposition logic is developed. In effect the system works as follows: The public encryption key consists of two statements p_0 and p_1 of proposition calculus. The secret decryption key is some truth assignment for the variables in p_0 and p_1. The assignment makes p_0 false and p_1 true. The encryption of bit i is done by shuffling the statement p_i into an equivalent but different looking statement p_i', which is the cryptotext. Decryption is done simply by computing the value of p_i' with the secret truth assignment.

Let us look at the details.

Definition 1. The set P_X of proposition statements over the finite variable set X is the smallest set such that

(1) $T,F \in P_X$,

(2) $X \subseteq P_X$ and

(3) if $p,q \in P_X$ then $(\neg p)$, $(p \vee q)$ and $(p \wedge q) \in P_X$.

Every mapping $\alpha : X \to \{T,F\}$ is called a truth assignment of the variables in X. The mapping α is in the traditional way extended into a mapping $\hat{\alpha} : P_X \to \{T,F\}$, which gives truth values for proposition statements over X.

In practice unnecessary parentheses are omitted using the normal precedences of connectives. The cryptosystem can now be defined.

Definition 2. Let X and Y be disjoint variable sets, α a truth assignment of X, and p_0 and p_1 statements in $P_{X \cup Y}$ such that $\hat{\beta}(p_0)=F$ and $\hat{\beta}(p_1)=T$ (or vice versa) for every truth assignment $\beta : X \cup Y \to \{T,F\}$ with $\beta(x)=\alpha(x)$ for each $x \in X$. So, if the variables of X get their truth values from α, then the values of the variables of Y have no effect on the values of p_0 and p_1.

The public encryption key consists of p_0 and p_1, while α constitutes the secret decryption key. The encryption of bit i is done by selecting an arbitrary truth assignment δ of Y and substituting in p_i every $y \in Y$ with $\delta(y)$. After that the result is reduced and shuffled using the rules below until it is no longer recognizable. The rules are given in the form $p \to q$ which tells that any occurence of p may be replaced by q. In the rules p,q and r mean arbitrary statements.

$\neg T \to F$	$\neg F \to T$
$(T \vee p) \to T$	$(F \vee p) \to p$
$(T \wedge p) \to p$	$(F \wedge p) \to F$
$(p \vee q) \to (q \vee p)$	$(p \wedge q) \to (q \wedge p)$
$(p \vee (q \vee r)) \to ((p \vee q) \vee r)$	$(p \wedge (q \wedge r)) \to ((p \wedge q) \wedge r)$
$(p \vee p) \to p$	$(p \wedge p) \to p$
$(p \vee \neg p) \to T$	$(p \wedge \neg p) \to F$

The statement $p \varepsilon P_X$ obtained after the shuffling is the cryptotext. Obviously it is equivalent to p_i under the truth assignment α. Consequently, the decryption can be done by computing $\hat{\alpha}(p)$.

The important question about how to construct keys p_0, p_1 and α is not considered here in detail. We just mention one possible approach. The variable sets X and Y are fixed and an arbitrary assignment $\alpha : X \to \{T,F\}$ is selected. Then statements p_0' and p_1' over the variable set Y are formed such that p_0' is a contradiction and p_1' is a tautology. This may be done with the help of backward resolution, for instance. One should make sure that p_0' and p_1' include many occurrences of symbols T and F. Now, every T in p_0' and p_1' is replaced with some variable $x \varepsilon X$ such that $\alpha(x)=T$. Similarly we get rid of occurrences of F. The resulting statements over $X \cup Y$ are the public keys p_0 and p_1.

In the following statement we make use of an ORACLE that is able to break our cryptosystem. When the ORACLE is called with a legal instance for the cryptanalysisproblem (that is, with public keys p_0 and p_1, and a possible cryptotext p) it returns the encrypted bit. With an illegal instance the ORACLE may work arbitrarily.

Theorem 2. Suppose L_0 and L_1 are languages in NP. There is a deterministic polynomial time algorithm using ORACLE that, when given w in $L_0 \Delta L_1$ ($= (L_0 \setminus L_1) \cup (L_1 \setminus L_0)$), determines whether $w \varepsilon L_0$ or $w \varepsilon L_1$.

Proof. Because the satisfiability problem of propositional logic is NP-complete, statements q_0^X, $q_1^X \varepsilon P_X$ can be constructed in polynomial time such that q_0^X is satisfiable iff $w \varepsilon L_0$ and q_1^X is satisfiable iff $w \varepsilon L_1$. Since w is in $L_0 \Delta L_1$, there exists an assignment $\alpha : X \to \{T,F\}$ that makes either q_0^X or q_1^X true, while the other is a contradiction.

Let Y be another variable set such that there is a bijection $f : X \to Y$. Construct new statements q_0^Y, $q_1^Y \in P_Y$ by replacing variables in q_0^X and q_1^X with their f-images.

Statements $(q_0^X \wedge \neg q_1^Y)$ and $(q_1^X \wedge \neg q_0^Y)$ constitute a legal encryption key with the assignment α as the secret decryption key. Namely, suppose without loss of generality that q_0^X is a contradiction and $\hat{\alpha}(q_1^X)=T$. Then regardless what values the variables of Y are given the statement $(q_0^X \wedge \neg q_1^Y)$ gets value F and $(q_1^X \wedge \neg q_0^Y)$ value T when the variables of X obtain their values from the assignment α.

The trivial statement F is a possible cryptotext. If the assignment δ used in encryption is accidently chosen so that each $y \in Y$ gets the value $\alpha(f^{-1}(y))$, then $\hat{\delta}(\neg q_1^Y)=F$ (still under the assumption $\hat{\alpha}(q_1^X)=T$). This obviously implies that $\neg q_1^Y$ where variables of Y have been replaced with their truth values can be reduced to F with the rules of definition 2, and thus $(q_0^X \wedge \neg q_1^Y)$ can be reduced first into $(q_0^X \wedge F)$ and further into F.

Finally ORACLE is consulted with an instance consisting of $p_0=(q_1^X \wedge \neg q_0^Y)$, $p_1=(q_0^X \wedge \neg q_1^Y)$ and $p=F$. Clearly $w \in L_i$ where i is the answer of ORACLE. □

Suppose we have any cryptosystem in the sense of section 2. According to theorem 1 there are languages L_0 and L_1 in NP that separate sets BIT_0 and BIT_1 of the system. Theorem 2 with these L_0 and L_1 says that there is a polynomial time algorithm using ORACLE that finds the encrypted bit, when given a legal instance (e,c). So, our cryptosystem is optimal in the sense that cryptanalysisproblem of any other system can be reduced to its cryptanalysis.

Another corollary of theorem 2 says that cryptanalysis of our system is NP∩CoNP-hard. If L is a language in NP∩CoNP, theorem 2 may be applied with $L_0=L$ and $L_1=\Sigma^* \setminus L$. The theorem says that there is a polynomial time algorithm using ORACLE that solves the membership problem of L.

5. CONCLUSIONS

In section 4 a cryptosystem based on proposition calculus was presented. It was proved secure in the sense that the cryptanalysis problem encountered by an eavesdropper is NP∩CoNP-hard. Moreover, the system was showed optimal in the sense that any cryptanalytic method against it can be used

to break other cryptosystems as well.

This seems quite excellent. But a closer inspection of the proof of theorem 2 makes one suspicious. A similar argument shows namely equivalent results for the following degenerate system:

The public key e consists of two statements p_0 and p_1 in proposition calculus. They are constructed so that one of them is a contradiction while the other is satisfiable. The secret key d equals k where p_k is the satisfiable statement.

The encryption of a bit i is done as follows: First select an arbitrary truth assignment for variables existing in p_i and compute its value under this assignment. If the result is T send a special mark *, else send the bit i itself.

The decryption algorithm simply changes * to k and leaves 0 and 1 unaltered.

This is indeed a cryptosystem in the sense presented in section 2. It is trivial to prove a theorem similar to theorem 2. Yet no one can seriously claim that this system is safe: If there are many truth assignments that make p_k true then the cryptanalyst can simply generate assignments until either p_0 or p_1 comes true, and so decide the meaning of *. On the other hand, if p_k is rarely true then the symbol * is not likely to occur as the cryptotext. (In general, no system with a small cryptotext space C can be secure.)

Unfortunately theorem 2 does not say anything more about the system of section 4 than it says about this degenerate system. The problem is apparent: The complexity theory as we used it deals with worst-case complexities - it says nothing about the complexity on the average. In a more accurate measure of safety the probabilities of different cryptotexts should be taken into account.

REFERENCES

(1) W. Diffie and M. Hellman, New directions in cryptography, IEEE Transactions on Information Theory IT-22 (1976) 644-654.

(2) S. Goldwasser and S. Micali, Probabilistic encryption & how to play mental poker keeping secret all partial information, in: Proceedings of the 14th ACM Symposium on the Theory of Computing (1982), 365-377.

(3) A. Salomaa, Computation and Automata (Cambridge University Press,1985).

IMYCS'88, Smolenice Castle, November 14-18, 1988.

A NOTE ON THE COMPUTATIONAL COMPLEXITY OF BRACKETING AND RELATED PROBLEMS*

MIRKO KŘIVÁNEK†

Abstract. It is shown that the problem of finding the minimum number of bracketing transfers in order to transform one bracketing to another bracketing is an *NP*-complete problem. This problem is related to problems on random walks and to the problem of a comparison of two (labeled) rooted trees. The latter problem is studied with connection to cluster analysis. Finally, one polynomially solvable class of bracketing problems is obtained.

I. Introduction and background. Bracketing problems have a long history [8]. Though the main emphasis was mainly concentrated on enumeration problems we shall be interested in the computational complexity of evaluation of the distance between two given bracketings. Finally, using the concept of closed random walks we shall stress the connection of bracketing problems to the problem of comparison and evaluation of two labeled rooted trees. This type of problem is often investigated in cluster analysis [6].

More precisely, the word w in the alphabet $\Sigma = \{(,)\}$ is said to be a *bracketing* if it is generated by the following rules :

$$S \longrightarrow SS|(S)|\Lambda,$$

where Λ stands for an empty word. The set of all bracketings is often called the Dycklanguage and plays an important part in the theory of formal languages [3]. The abbreviation $l^i, i > 0, l \in \Sigma$, denotes $\underbrace{l \ldots l}_{i \text{ times}}$. Let $\mathcal{B}$ ($\mathcal{B}_n$, resp.) be a set of all bracketings over Σ (...of length n, resp.). Note that n is even. A bracketing $b' \in \mathcal{B}$ is said to be a *sub-bracketing* of b, written $b' \subset b$, if b' is a proper subword of b. The *nesting level* of a sub-bracketing b' of b is the number of different sub-bracketings of b which contain b' as their sub-bracketing. Given two bracketings $b_1, b_2 \in \mathcal{B}_n$ we say that bracketing b_2 arises from bracketing b_1 by one *bracketing transfer* if there is a sub-bracketing b of b_2 such that

$$b_1 = xby \quad \text{and either} \quad b_2 = x_1bx_2y, \quad \text{where} \quad x_1x_2 = x,$$

$$\text{or} \quad b_2 = xy_1by_2, \quad \text{where} \quad y_1y_2 = y, \quad \text{for} \quad x, x_2, y, y_1 \in \Sigma^+, x_1, y_2 \in \Sigma^*.$$

By $\beta(b_1, b_2)$ the minimum number of bracketing transfers needed to transform b_1 to b_2 will be denoted, i.e.

$\beta(b_1, b_2) = j$ if there is a sequence $s_1, s_2, \ldots, s_{j+1}$ of bracketings from $\mathcal{B}_n$ such that

$$b_1 \equiv s_1, b_2 \equiv s_j, \beta(s_i, s_{i+1}) = 1 \quad \text{for} \quad i = 1, \ldots, j.$$

First we have the following straightforward lemma :

---

* Extended abstract

† Institute for Mathematics and Its Applications, University of Minnesota and Charles University, Prague.

LEMMA 1. *The function β is the distance measure on $\mathcal{B}_n$ and $(\mathcal{B}_n, \beta)$ is a metric space.* □

The underlying computational problem **BR** in which lies our main interest is stated as follows :

INSTANCE : Two bracketings $b_1, b_2 \in \mathcal{B}_n$, positive integer k ;

QUESTION : Does it hold that $\beta(b_1, b_2) \leq k$?

Our NP-completeness terminology is that of [2].

II. Complexity results. First we shall prove the following.

THEOREM 1.

The problem **BR** *is NP-complete.*

Proof. Clearly, the problem **BR** is in the class NP. We shall exhibit a polynomial transformation from the problem **BIN PACKING** which is known to be strongly NP-complete [2]. **BIN PACKING** has been introduced as follows :

INSTANCE : Positive integers $i_1, \ldots, i_s, B, r$ such that $\sum_{j=1}^{s} i_j = rB$;

QUESTION : Is there a partition of $\{i_1, \ldots, i_s\}$ into r classes $I_1, \ldots, I_r$ such that $\sum_{j \in I_m} i_j = B, m = 1, \ldots, r$?

Given an instance of **BIN PACKING** the instance of the problem **BR** is constructed in polynomial time by putting

$$b_1 \stackrel{def}{=} \underbrace{(^B)^B \ldots (^B)^B}_{r \text{ times}}, \quad b_2 \stackrel{def}{=} (^{i_1})^{i_1} (^{i_2})^{i_2} \ldots (^{i_s})^{i_s}, \quad k \stackrel{def}{=} s - r.$$

Now, the equivalence

$$\beta(b_1, b_2) = s - r \Leftrightarrow \textbf{BIN PACKING} \text{ has "yes"-solution}$$

is easily verified and the theorem is proved. □

Theorem 1 says that it is very unlikely that there exists a polynomial algorithm for the problem **BR**. Therefore we would like to exhibit a polynomial approximation for the problem **BR**. Notice that proof of Theorem 1 does not exclude the possible existence of such an algorithm. The so-called "next fit" approximation algorithm has been believed to provide a "good" polynomial approximation for **BR** since **BR** generalizes in some way the **BIN PACKING** problem. Recall that the next fit algorithm was proved to be a "good" approximation for **BIN PACKING** both from the worst and average case complexity viewpoint [2,4]. Formally the approximation algorithm $\mathcal{A}$ for **BR** is encoded as follows :

Algorithm $\mathcal{A}$:

(Step 1.) $s_1 := b_1$; $s_2 := b_2$;

(Step 2.) **do** $2n$ **times**

Scan and compare current letters of s_1 and s_2;

if they are different **then**

{*suppose that scanned letter in* s_1 *is "(" , i.e.* $s_1 = x($, $x, y \in \Sigma^*$}

(Step 3.) find in s_2 a "next" sub-bracketing b (minimal/maximal with respect to the current nesting level) such that $s_2 = x)y_1 b y_2$, $y_1, y_2 \in \Sigma^*$;

$s_2 := xb)y_1 y_2$;

endif

endo

endalgorithm

The correctness and time analysis of the algorithm $\mathcal{A}$ is established in the following theorem :

THEOREM 2. *Algorithm* $\mathcal{A}$ *runs in polynomial time and solves problem* **BR** *using* $O(n)$ *bracketing transfers.*

Proof. Rough time estimate for Step 3 is $O(n)$. This yields $O(n^2)$ time complexity of the algorithm $\mathcal{A}$. The algorithm $\mathcal{A}$ transforms bracketing b_1 into bracketing b_2. This is observed from the fact that eventually both words s_1, s_2 produced by $\mathcal{A}$ are equal. As possibly both b_1, b_2 and consequently s_1, s_2 are changed the sequences $b_1 \longrightarrow s_1$ and $s_2 \longrightarrow b_2$ provide the sequence of bracketing transfers required for transforming b_1 to b_2. In the worst case the number of bracketing transfers is proportional to the corresponding number of sub-bracketings of b_1 (b_2, resp.) and thus it is $O(n)$. □

Remark. Using so-called search trees [7] as a data structure for the representation of bracketings, Step 3 can be implemented in $O(\log n)$ time. Asymptotically $O(n \log n)$ upper bound is the best possible for polynomially solvable instances of **BR** since the well-known **SORTING** problem is linearly transformable (assuming an unary representation of input numbers) to the following instance of **BR** :

$$b_1 = ()(()) \ldots ({}^{n})^{n}, b_2 = ({}^{x_1})^{x_1}({}^{x_2})^{x_2} \ldots ({}^{x_n})^{x_n} \quad \text{where} \quad \{x_1, x_2, \ldots, x_n\} = \{1, \ldots, n\}$$

constitute an instance of **SORTING**. Recall that **SORTING** is solvable in $\Theta(n \log n)$ time [7]. □

Let us deal with the question of how good the approximation produced by $\mathcal{A}$ is. Regrettably no constant bounded worst case error ratio is guaranteed.

THEOREM 3. *Algorithm $\mathcal{A}$ has $\Theta(n)$ worst case error ratio.*

Proof. Let $b_1 = xy, b_2 = yx, x = (), y = ()(\underbrace{()\dots()}_{O(n) \text{ times}})$. In this case $\beta(b_1, b_2) = 1$. However algorithm $\mathcal{A}$ constructs a sequence of $O(n)$ bracketing transfers regardless of the nesting level of the "next" sub-bracketing in Step 3. □

The failure of algorithm $\mathcal{A}$ is due to the fact that $\mathcal{A}$ does not search for identical sub-bracketings in b_1 and b_2. Therefore its behavior could be slightly improved by pre-processing, i.e. by decomposing b_1 and b_2 into their corresponding sub-bracketings, say, maximal up to inclusion. This approach supposes setting up a data structure where nesting level and sub-bracketing can be directly accessed. This way we avoid pathological behavior of $\mathcal{A}$ on the current nesting level but complexity problems remain unchanged when dealing with identical sub-bracketings on different nesting levels.

Let us conclude this section by a remark that some preliminary calculations indicate that algorithm $\mathcal{A}$ also has the average case error ratio of order $\Theta(n)$. The details will appear in a full paper.

III. Random walks and rooted trees. The aim of this section is to discuss a 1-1 correspondence between bracketings, random walks and rooted trees. It will enable us to extend the results of the previous section to trees embedded to the plane. Our exposition is based on [5].

Random walk of length n is a $(n+1)$-tuple $\Phi = (\varphi(0), \varphi(1), \dots \varphi(n))$ where φ is a mapping to non-negative integers such that

$$\varphi(0) = \varphi(n) = 0, \varphi(i) \in \{\varphi(i-1) - 1, \varphi(i-1) + 1)\}, i = 1, \dots, n.$$

LEMMA 2. *There is a one-to-one correspondence between $\mathcal{B}_n$ and the set of all random walks of length n.*

Proof. Let $b \in \mathcal{B}_n$. Define a random walk Φ of length n as follows

$$\varphi(i) = \begin{cases} 0, & \text{if } i \in \{0, n\} \\ \varphi(i-1) + 1, & \text{if } i\text{-th letter of } b \text{ is "("} \\ \varphi(i-1) - 1, & \text{if } i\text{-th letter of } b \text{ is ")"}. \end{cases} \quad \square$$

Let T be a rooted tree on n vertices embedded into the plane. Let us consider a topological ordering $\omega_T = v_0 v_1 \dots v_{2n-2}$ of its vertices which is recursively defined as follows :

(1) If $T = \{v_0\}$ then $\omega_T = v_0$,

(2) If T has a root v_0 with the subtrees $T_1, T_2, \dots, T_k$ then $\omega_T = v_0 \omega_{T_1} v_0 \omega_{T_2} v_0 \dots \omega_{T_k} v_0$.

The set of all trees on n vertices with a root v_0 embedded into the plane will be denoted by $\mathcal{T}_n$.

Let $T_1, T_2 \in \mathcal{T}_n$. T_1 is said to be obtained from T_2 by one *subtree modification* if

$$T_1 \approx T_2 - uv + uw, \text{ where } uv \in E(T_2), uw \notin E(T_2),$$

where symbol $\approx$ expresses that both trees have the same topology of plane embedding, i.e. they are *topologically isomorphic*. The *subtree distance* $\tau(T_1, T_2)$ between T_1 and T_2 is defined as the minimum number of subtree modifications needed to be performed on T_2 in order to obtain a tree which is topologically isomorphic to T_1. Notice that the root v_0 is supposed to be fixed. The following observations are quite straightforward:

LEMMA 3. *The pair $(\mathcal{T}_n, \tau)$ forms a metric space.* □

LEMMA 4. *There is a one-to-one correspondence between $\mathcal{T}_n$ and random walks of length $2n-2$.*

Proof. Let $T \in \mathcal{T}_n$. Let $d(v_i, v_0)$ be the distance of the i-th vertex v_i of ω from v_0. The corresponding random walk $\Phi = (\varphi(0), \ldots, \varphi(2n-2))$ is defined as follows :

$$\varphi(i) = \begin{cases} 0, & \text{if } i \in \{0, 2n-2\} \\ d(v_i, v_0), & \text{otherwise.} \end{cases} \square$$

Now we are ready to prove the following :

THEOREM 4. *Given $T_1, T_2 \in \mathcal{T}_n$, the underlying decision problem of computing $\tau(T_1, T_2)$ is NP-complete.*

Proof. Choose $b_1, b_2 \in \mathcal{B}_n$ and by virtue of Lemma 4 consider two corresponding trees $T_1, T_2 \in \mathcal{T}_n$. By the aid of Lemma 2 and Lemma 4 we have

$$\beta(b_1, b_2) = \tau(T_1, T_2).$$

The use of Theorem 1 completes the proof. □

Trees from $\mathcal{T}_n$ are very often constructed by hierarchical clustering procedures. The study of the consensus between these trees is one of the most important problems encountered in cluster analysis [6]. However, special attention is mostly paid to binary trees [1]. The concept of random walks can be used for proving similar NP-completeness results for binary trees, too.

Let $T \in \mathcal{T}_{2n-1}$ be a binary rooted tree on n leaves, i.e. having all internal vertices of degree 3 except of the root v_0 which is of degree 2. A given binary rooted tree T induces a topological ordering $\omega_T = v_0 v_1 \ldots v_{2n-2}$ which is defined recursively as follows :

(1) If $T = \{v_0\}$ then $\omega_T = v_0$,

(2) If T has a root v_0 with the subtrees T_1, T_2 then $\omega_T = \omega_{T_1} \omega_{T_2} v_0$.

The set of all binary trees on n leaves and with the root v_0 embedded into the plane will be denoted by $\mathcal{T}_n^b$. Given $T_1, T_2 \in \mathcal{T}_n^b$ we say that T_1 is obtained by one *(binary) subtree modification* if

$$T_1 \approx T_2 - \{\{u_1, v\}, \{u_1, w\}\} + \{\{u_2, v\}, \{u_2, w\}\}$$

where

$$\{u_1, v\}, \{u_1, w\} \in E(T_2) \text{ and } \{u_2, v\}, \{u_2, w\} \notin E(T_2).$$

Notice that u_2 is a leaf. The *(binary) subtree distance* τ^b is defined as the minimum number of subtree modifications required to obtain a tree T_1 from T_2. Similarly as in the general case the following propositions hold

LEMMA 5. *The pair* $(\mathcal{T}_n^b, \tau^b)$ *forms a metric space.* □

LEMMA 6. *There is a one-to-one correspondence between binary rooted trees on* n *leaves and random walks of length* $2n-2$.

Proof. Let $T \in \mathcal{T}_n, \omega = v_0 v_1 \dots v_{2n-2}$. The corresponding random walk Φ is defined as follows

$$\varphi(i) = \begin{cases} 0, & \text{if } i \in \{0, 2n-2\} \\ \varphi(i-1)+1, & \text{if } v_i \text{ is a leaf in } T \\ \varphi(i-1)-1, & \text{if } v_i \text{ is an internal vertex in } T. \end{cases}$$ □

Combining Lemma 2, Lemma 6 and Theorem 1 we get

THEOREM 5. *Given* $T_1, T_2 \in \mathcal{T}_n^b$ *the underlying decision problem of computing* $\tau^b(T_1, T_2)$ *is NP-complete.* □

IV. Labeled rooted trees. In this section a polynomially solvable class of bracketing problems will be explored by means of labeled rooted trees. Let $T \in \mathcal{T}_n$ and let ω_T be its topological ordering. Let us define on the set of vertices of T a *labeling* ξ, $\xi : \{v_0, \dots, v_{n-1}\} \to \{0, \dots, n-1\}$ as follows :

$$\xi(x) = \begin{cases} 0, & \text{if } x \equiv v_0 \\ i, & \text{if vertex } x \text{ occurs as the } i\text{-th new vertex in } \omega_T. \end{cases}$$

Let us suppose that we are given a fixed labeling ξ on $\{v_0, \dots, v_{n-1}\}$. Let $\mathcal{T}_n^\xi$ denote the set of all labeled trees on n vertices with the root v_0 and with the same labeling ξ. Now we can define a subtree modification distance τ^ξ between labeled rooted trees from $\mathcal{T}_n^\xi$ formally in the same way as in the unlabeled case with the only exception that now the labeling ξ must be preserved by subtree modifications. Clearly Lemma 3 and Lemma 4 can be rewritten as follows :

LEMMA 7. *The pair* $(\mathcal{T}_n^\xi, \tau^\xi)$ *forms a metric space.* □

LEMMA 8. *There is a one-to-one correspondence between* $\mathcal{T}_n^\xi$ *and random walks of length* $2n-2$. □

Let us define two graphs

$$\mathcal{G}_1 = (\mathcal{T}_n, E_1), \quad \mathcal{G}_2 = (\mathcal{T}_n^\xi, E_2)$$

where

$$\{T_1, T_2\} \in E_1 \Leftrightarrow \tau(T_1, T_2) = 1 \quad \text{for} \quad T_1, T_2 \in \mathcal{T}_n,$$

$$\{T_1, T_2\} \in E_2 \Leftrightarrow \tau^\xi(T_1, T_2) = 1 \quad \text{for} \quad T_1, T_2 \in \mathcal{T}_n^\xi.$$

It is easy to see that $\mathcal{G}_2$ is a proper subgraph of $\mathcal{G}_1$. This observation justifies the following theorem :

THEOREM 6. *Given* $T_1, T_2 \in \mathcal{T}_n^\xi$, *the problem of the computation of* $\tau^\xi(T_1, T_2)$ *is polynomially solvable.*

Proof. Let us consider the following algorithm :

(Step 1.) **do** traverse the tree T_2 using so-called breath-first search [7]
(Step 2.) **if** childrens of the current vertex of T_1 and T_2 are different
then update locally the tree T_2 by T_1
endo

The loop involved in Step 1 requires $O(n)$ time, Step 2 can be implemented in $O(\log n)$ time using search trees as data structures for the fast search and update in T_1 and in T_2. □

It is left to the reader to find an example which shows that the algorithm outlined above has $\Theta(n)$ worst case error ratio if it is used as an approximation for a general bracketing problem.

REFERENCES

[1] K. ČULÍK II. AND D. WOOD, *A note on some tree similarity measures*, Information Processing Letters, 15 (1982), pp. 39–42.
[2] M.R. GAREY AND D.S. JOHNSON, *Computers and intractability*, W.H. Freeman, San Francisco, 1979.
[3] M.A. HARRISON, *Introduction to formal language theory*, Addison-Wesley, 1978.
[4] M. HOFRI, *Probabilistic analysis of algorithms*, Springer-Verlag, 1987.
[5] R. KEMP, *Fundamentals of the average case analysis of particular algorithms*, Wiley-Teubner, Stuttgart, 1984.
[6] M. KŘIVÁNEK, *The computational complexity of the consensus between hierarchical trees*, Proc. IMYCS' 84, Smolenice, pp. 119–125.
[7] K. MEHLHORN, *Data structures and algorithms*, Springer-Verlag, 1984.
[8] SCHRÖDER, *Vier Combinatorische Probleme*, Z. für M. Phys., 15 (1870), pp. 361–376.

IMYCS'88, Smolenice Castle, November 14-18, 1988.

CHOMSKY HIERARCHY AND COMMUNICATION COMPLEXITY HIERARCHY

GALINA KUMIČÁKOVÁ

Mathematical Institute
Slovac Academy of Sciences
Ždanovova 6, 040 01 Košice
Czechoslovakia

1. INTRODUCTION

The communication complexity for VLSI is a powerful tool for proving lower bounds on VLSI circuits. It is well-known that the communication complexity of any language L provides direct lower bound on the area complexity of VLSI circuits recognizing L, and that communication complexity squared provides direct lower bound on the complexity measure area.time squared (AT^2) of VLSI circuits.

Originally, the communication complexity was defined in Papadimitriou and Sipser [10] and investigated in several papers [2, 3, 5-9, 11]. Informally, it can be defined in the following way. Suppose a language $L \subseteq (\{0, 1\}^2)^*$ has to be recognized by two distant computers. Each computer recieves half of the input bits, and the computation proceeds using some protocol for communication between the two computers. The minimal number of bits that has to be exchanged in order to successfully recognize $L \cap \{0, 1\}^{2n}$, minimized over all partitions of the input bits into two equal parts, and

considered as a function of n, is called the communication complexity of L.

An improvement of the communication complexity model was suggested in Aho at al. [1] and formally defined and investigated as so called S-communication complexity in Hromkovič [4]. Since the model of S-communication complexity provides useful lower bounds on A and AT^2 of VLSI circuits in many cases in which the original communication complexity provides no reasonable lower bounds (see [4] for details) we study the S-communication complexity in this paper.

The relation between communication complexity hierarchy and Chomsky hierarchy was studied in [7]. S-communication complexity differs essentially from the communication complexity in the sense that the results from [7] cannot be directly transformed in order to obtain the relation between S-communication coplexity hierarchy and Chomsky hierarchy. In this paper we show that all regular languages are recognazible within constant S-communication complexity but, for any constant c, there is a regular language which cannot be recognized within S-communication complexity less than c. Next we show that there are hard languages according to the Chomsky hierarchy (e.g. outside the class of recursively enumerable languages) which can be recognized within one bit S-communication complexity. The main result is the linear lower bound on S-communication complexity of deterministic context-free language. It implies that VLSI circuits need $\Omega(n)$ area and $\Omega(n^2)$ area.time squared complexity to recognize deterministic context-free language which solves an open problem of Hromkovič [7].

The structure of the paper is as follows. In Section 2 the definitions and notations are given. The relation between the Chomsky hierarchy and the S-communication complexity hierarchy is studied in Section 3.

2. DEFINITIONS AND NOTATIONS

Now, let us formally define the S-communication complexity

in the same way as in [4]. In the paper N denotes the set of all natural numbers (positive integers). For $n \in N$, let $[n] = \{1, 2, \ldots, n\}$ and let $|A|$, for a finite set A, denote the number of elements in A.

Let Y be a subset of $[n]$ such that $|Y| = 2m$, for some m. A partition of $[n]$ according to Y is a pair $\pi = (S_I, S_{II})$, where $S_I \cap S_{II} = \emptyset$, $S_I \cup S_{II} = [n]$ and $|S_I \cap Y| = |S_{II} \cap Y| = m$. We denote by x_I (x_{II}) the input word x from $\{0, 1\}^n$ restricted to the set S_I (S_{II}), and we write $x = \pi^{-1}(x_I, x_{II})$.

A protocol on n inputs is a triniply $D_n = (Y, \pi, \Phi)$, where

(a) Y is a subset of $[n]$ and $|Y| = 2m$, for some m,

(b) $\pi = (S_I, S_{II})$ is a partition of $[n]$ according to Y (which corresponds to the partition of the input bits for two computers),

(c) Φ is a function from $(\{0, 1\}^{|S_I|} \cup \{0, 1\}^{|S_{II}|}) \times \{0, 1, \$\}^*$ to $\{0, 1\}^* \cup \{\text{accept}, \text{reject}\}$ which has the following prefix-freeness property: for a given string $c \in \{0, 1, \$\}^*$ and two different $y, y' \in \{0, 1\}^{|S_I|}$ ($\{0, 1\}^{|S_{II}|}$) $\Phi(y, c)$ is not a proper prefix of $\Phi(y', c)$. (Intuitively, Φ describes the communication between the two computers.)

A computation of D_n on input word x in $\{0, 1\}^n$ is a string $c = c_1\$c_2\$\ldots c_k\$c_{k+1}$, where $k \geq 0$, $c_1, \ldots, c_k \in \{0, 1\}^*$ $c_{k+1} \in \{\text{accept}, \text{reject}\}$, and such that for each integer j, $0 \leq j \leq k$, we have

(1) if j is even, then $c_{j+1} = \Phi(x_I, c_1\$c_2\$\ldots\$c_j)$, and

(2) if j is odd, then $c_{j+1} = \Phi(x_{II}, c_1\$c_2\$\ldots\$c_j)$.

The computation $c = c_1\$c_2\$\ldots\$c_{k+1}$ is called accepting iff c_{k+1} = accept and the length of this computation is the total length of all messages c_j, $1 \leq j \leq k$ (ignoring $\$'s and final accept/reject).

The S-communication complexity of $D_n = (Y, \pi, \Phi)$, shortly $SC(Y, \pi, \Phi)$, is the maximum of the lengths of all computations of D_n.

We say that D_n computes the Boolean function $h: \{0, 1\}^n \to \{0, 1\}$ (within S-communication complexity c) if, for each $x \in \{0, 1\}^n$, there exists the computation of D_n on input word x (of length at most c), and it is accepting iff $h(x) = 1$.

The S-communication complexity of h according to Y is defined as $SC(h, Y) = \min\{SC(Y, \pi, \Phi) \mid$ the protocol (Y, π, Φ) computes $h\}$, and the S-communication complexity of h as $SC(h) = \max\{SC(h, Y) \mid Y \subseteq [n]$ and $|Y| = 2m$ for some $m\}$. Obviously, for each Boolean function $h: \{0, 1\}^n \to \{0, 1\}$, $SC(h) \leq n/2$ holds (in fact, the first computer may send all its input bits to the second one).

Let $L \subseteq \{0, 1\}^*$ be a language. The S-communication complexity of L is defined as a function SC_L from N to non-negative integers such that $SC_L(n) = SC(h_n^L)$, where $h_n^L: \{0, 1\}^n \to \{0, 1\}$, and $h_n^L(x) = 1$ iff $x \in L \cap \{0, 1\}^n$.

Let f be a real function defined on natural numbers. We say that L is recognizable within S-communication complexity f(n), shortly $L \in SCOMM(f(n))$, if $SC_L(n) \leq f(n)$ holds for any natural n. Obviously, each language is recognizable within S-communication complexity n/2. In [4] it is shown that, for all functions $f: N \to N$ such that $1 \leq f(n) \leq n/2$, $SCOMM(f(n)-1) \subsetneq SCOMM(f(n))$. So, the hierarchy of languages according to S-communication complexity is obtained.

Before starting our study, we introduce some notation. For a word w, $\#_a(w)$ denotes the number of occurences of the symbol a in w, for a real number x, $\lfloor x \rfloor$ ($\lceil x \rceil$) is the floor (ceiling) of x, and for a nonnegative integer i, $BIN_j(i)$ is the binary code of i on j bits (e.g. $BIN_4(5) = 0101$).

3. CHOMSKY HIERARCHY AND S-COMMUNICATION COMPLEXITY

First, we study the S-communication complexity of regular languages.

Theorem 1. For each regular language $L \subseteq \{0, 1\}^*$ there exists a constant c such that $L \in SCOMM(c)$.

P r o o f. Let $L \subseteq \{0, 1\}^*$ be a regular language. Then there exists a deterministic finite automaton A recognizing L and having s states $q_0, q_1, \ldots, q_{s-1}$, for some s. We show that $L \in SCOMM(c)$, where $c = \lceil \log_2 s \rceil$.

Let n be a natural number and let $Y = \{i_1, i_2, \ldots, i_{2m}\}$, where $i_1 < i_2 < \ldots < i_{2m}$, be a subset of $[n]$. We consider the protocol $D_n = (Y, \pi, \Phi)$, where $\pi = (S_I, S_{II})$ is the partition of $[n]$ according to Y such that $S_I = [i_m]$ and for all x in $\{0, 1\}^{i_m}$, y in $\{0, 1\}^{n-i_m}$, j in $\{0, 1, \ldots, s-1\}$

$\Phi(x, \varepsilon) = BIN_c(i)$ iff A computing on x ends in the state q_i,

$\Phi(y, BIN_c(j))$ = accept (reject) iff A beginning to compute on y in the state q_j ends the computation in an accepting (unaccepting) state.

It is easy to see that the protocol $D_n = (Y, \pi, \Phi)$ computes function h_n^L within S-communication complexity c. □

Theorem 2. For all natural c there exists a regular language L such that L does not belong to SCOMM(c).

P r o o f. Let c be a natural number and let us consider the regular language $L = \{x \in \{0, 1\}^* \mid \#_1(x) = 2^{c+1}\}$. We prove by contradiction that, for $n = 2^{c+2}$ and $\bar{Y} = [n]$, $SC(h_n^L, \bar{Y}) > c$ holds, which is sufficed to prove the assertion.

Let there exists the protocol $D_n = (\bar{Y}, \pi, \Phi)$, $\pi = (S_I, S_{II})$, $n = 2^{c+2}$, computing h_n^L within S-communication complexity c. Then the number of all accepting computations of D_n is at most $2^{c+1} - 1$ (because of the prefix-freeness property of Φ). Consider $2^{c+1} + 1$ disjoint nonempty classes $L_i = \{x \in \{0, 1\}^n \mid \#_1(x_I) = i$ and $\#_1(x_{II}) = 2^{c+1} - i\}$, $i = 0, 1, \ldots, 2^{c+1}$, of the words from L. There exist two input words x in L_i and y in L_j, $i \neq j$, having the same accepting computation. Then the computation of D_n on input word $z = \pi^{-1}(x_I, y_{II})$ is the same, i.e. accepting, although z does not belong to L. So, we have a contradiction. □

We obtained that all regular languages can be recognized within constant S-communication complexity but there is no constant c such that the family of all regular languages is included in SCOMM(c). Next theorem shows that much more languages belong to SCOMM(1).

Theorem 3. Each language $L \subseteq \{0, 1\}^*$ such that L involves at most one word of length n, for all n, belongs to SCOMM(1).

P r o o f. Let $L \cap \{0, 1\}^n$ involves the word $\bar{x}$. For any $Y \subseteq [n]$, $|Y| = 2m$, and a partition (S_I, S_{II}) of $[n]$ according to Y, we can informally describe the function Φ as follows. The first computer rejects in the case that its input $x \in \{0, 1\}^{|S_I|}$ disagrees with $\bar{x}_I$. If the input x agrees with $\bar{x}_I$, it sends the message 1 to the second computer which accepts (rejects) if its input agrees (disagrees) with $\bar{x}_{II}$. □

Corollary 1. Each language $L \subseteq \{0\}^*$ ($\{1\}^*$) belongs to SCOMM(1). □

Corollary 2. There exists a language which is not recursively enumerable and belongs to SCOMM(1). □

Considering Theorem 3 and its corollaries we obtain that there are hard languages according to the Chomsky hierarchy which are recognizable within S-communication complexity 1. Further, we solve an open problem of Hromkovič [7] proving a linear lower bound on S-communication complexity of a deterministic context-free language.

Theorem 4. There exists a deterministic context-free language L such that $SC_L(n) \geq n/32 - 1/2$, for all sufficiently large natural numbers n.

P r o o f. Let us consider the following deterministic context-free language

$$L = \{0w_1 0w_2 0 \ldots 0w_{a-1} 0w_a 1^b 0w_a 0w_{a-1} 0 \ldots 0w_2 0w_1 1^c \mid a, b, c \geq 1, w_i \in \{0, 1\} \text{ for } i = 1, 2, \ldots, a\}.$$

Let n be sufficiently large natural number, let r be even one of the numbers $\lfloor n/8 \rfloor$, $\lfloor n/8 \rfloor - 1$ (i.e. $8r \leq n$) and let $\bar{Y}$ denote the set $\{2, 4, 6, \ldots, 2r\}$ of size r. A set $\{i, j\}$, $i, j \in [n]$, $i \neq j$, is said to be divided by a partition $\pi = (S_I, S_{II})$ of $[n]$ iff neither S_I nor S_{II} involves both i and j. First, we will formulate a Lemma, then considering this assertion we will prove Theorem 4, and finally the Lemma will be proved.

Lemma. For any partition π of $[n]$ according to $\bar{Y}$ there exists a natural number m, $2r < m < 6r$, such that at least r/4 of the sets $\{2i, m+2r-2i+2\}$, $i = 1, 2, \ldots, r$, are divided by the partition π.

Now, we show by contradiction that the S-communication complexity of each protocol $D_n = (\bar{Y}, \pi, \Phi)$ computing h_n^L is at least r/4, which proves Theorem 4 ($r/4 \geq n/32 - 1/2$). So, let $D_n = (\bar{Y}, \pi, \Phi)$ be a protocol computing h_n^L within S-communication complexity $k - 1 < r/4$, $k \in N$. Then there exists at most $2^k - 1$ accepting computations of this protocol on the input words $x \in \{0, 1\}^n$. According to Lemma there exists m, $2r < m < 6r$, and there exist $i_1, i_2, \ldots, i_k$ such that, for all $j = 1, 2, \ldots, k$, i_j is even, $2 \leq i_j \leq 2r$ and the set $\{i_j, m+2r-i_j+2\}$ is divided by π.

Let $z^s = z_1^s z_2^s \ldots z_k^s$, for $s = 1, 2, \ldots, 2^k$, be all words from $\{0, 1\}^k$. For each s in $[2^k]$ the following word $y^s = y_1^s y_2^s \ldots y_n^s$ belongs to L

$y_{i_j}^s = y_{m+2r-i_j+2}^s = z_j^s$, for all $j = 1, 2, \ldots, k$,

$y_t^s = 1$, for all natural t such that $2r < t \leq m$ or $m+2r < t \leq n$,

$y_t^s = 0$, otherwise.

There exist s and s', $s \neq s'$, such that y^s and $y^{s'}$ have the same accepting computation of D_n and so, the computation of D_n on the word $\pi^{-1}(y_I^s, y_{II}^{s'})$ is also accepting. It is not difficult to see that this word does not belong to L.

P r o o f of Lemma. Let π be a partition of $[n]$ according to $\bar{Y}$. Let us consider the r x r matrix $A = \|a_{ij}\|$ such that, for all $i, j = 1, 2, \ldots, r$, $a_{ij} = 1$ (0) iff the set $\{2i, 6r-2j+2\}$ is (not) divided by π. We note that exactly r/2 elements from $\{2, 4, 6, \ldots, 2r\} = \bar{Y}$ belong to S_I (S_{II}) and so, exactly $r^2/2$ elements of matrix A are equal to 1. Consider the partition of elements of A into 2r - 1 following sets

$A_q = \{a_{ii+q} \mid i \in [r-q]\}$, for all $q = 0, 1, \ldots, r-1$,

$A_q = \{a_{ii+q} \mid i \in \{1-q, 2-q, \ldots, r\}\}$, for all $q = -1, -2, \ldots, -(r-1)$.

There must exist an integer p, $-(r-1) \leq p \leq r-1$, such that at least r/4 elements from A_p are equal to 1, i.e. there exist at least r/4 subscripts i from [r] such that the set $\{2i, 6r-2p-2i+2\}$ is divided by π. Thus, we can take $m = 4r-2p$. □

Corollary. There is a deterministic context-free language requiring linear area and $AT^2 = \Omega(n^2)$ to be recognized on any VLSI circuit. □

We conclude this paper with the note that there are the hard (simple) languages according to the Chomsky hierarchy which are simple (hard) according to the S-communication complexity hierarchy and so, there is no substantial coherence between these hierarchies of languages.

REFERENCES

[1] A.V. Aho, J.D. Ullman and M. Yannakakis, On notions of information transfer in VLSI circuits, Proc. 15th Ann. ACM Symp. on Theory of Computing (ACM 1983) 133-139.

[2] P. Ďuriš, Z. Galil and G. Schnitger, Lower bounds on communication complexity, Inform.and Comput. 73, 1(1987) 1-22.

[3] S. Hornick and M. Sarrafzatch, On problems transformability in VLSI, Algorithmica 1(1987) 97-112.

[4] J. Hromkovič, A new approach to defining the communication complexity for VLSI, Proc. 12th MFCS 86 (Lecture Notes in Computer science 233, Springer-Verlag, Berlin-Heildelberg-New York 1986) 431-439.

[5] J. Hromkovič, Communication complexity hierarchy, Theoret. Comput. Sci. 48(1986) 109-115.

[6] J. Hromkovič, Lower bound techniques for VLSI algorithms, Proc. IMYCS'86 (Hungarian Academy of Sciences, 1986) 9-19.

[7] J. Hromkovič, Relation between Chomsky hierarchy and communication complexity hierarchy, Acta Math. Univ. Comenian. XLVIII-XLIX (1986) 311-317.

[8] J. Ja´Ja and V.K. Prasanna Kumar, Information transfer in distributed computing with applications to VLSI, J. Assoc. Comput. Mach. 31(1984) 150-162.

[9] J. Ja´Ja, V.K. Prasanna Kumar and J. Simon, Information transfer under different sets of protocols, SIAM J. Comput. 13(1984) 840-849.

[10] Ch. Papadimitriou and M. Sipser, Communication complexity, Proc. 14th Ann. ACM Symp. on Theory of Computing (ACM 1982) 196-200.

[11] A.C. Yao, The entropic limitations of VLSI computations, Proc. 13th Ann. ACM Symp. on Theory of Computing (ACM 1981) 308-311.

IMYCS'88, Smolenice Castle, November 14-18, 1988.

LOWER BOUND FOR LINEAR SYSTOLIC ARRAYS

Dana Pardubská

Department of Theoretical Cybernetics
Comenius University
842 15 Bratislava
Czechoslovakia

1. Introduction

We consider a special type of one dimensional systolic arrays that belong to most studied VLSI computation models [1,2,4 - 7] . The "special type" means that we have only one input/output processor, fixed timing of input and real-time computation. This model, called two-way linear systolic automaton, has been shown to be quite powerful. For example in [3] it is shown that the languages coding symmetric, monotone, linear and self-dual Boolean functions can be recognized by this device.

An open problem has appeared - to find a specific language that cannot be recognized by two-way linear systolic automaton. The communication complexity approach providing the lower bound technique for VLSI cannot help in this case. The

bestknown simulation of f(n)-time linear systolic array by Turing machines is $f^2(n)$. So the lower bounds obtained for Turing machines till now do not provide any lower bound for two-way linear systolic automaton.

We develop a simple technique that enables us to prove that the specific language cannot be recognized on linear systolic arrays.

The two-way linear systolic automaton is a linear array of processors with two-way communication and one input/ output processor (see fig.1.). The processors are working simultaneously in discrete time. The input word $w = a_1 a_2 \dots a_n$ is inputed into the automaton one symbol after the other; the i-th symbol a_i in the time unit i. The automaton is considered to work in real time, i.e. output is obtained in the time unit n+1.

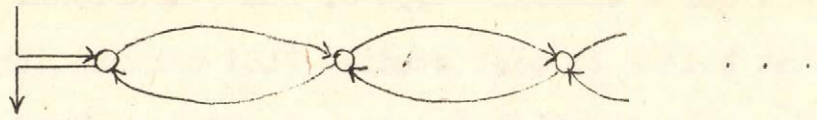


fig.1.

Definition A two-way linear systolic automaton 2LS is a 5-tuple $A = (T, \Sigma, \Gamma, \delta, \Gamma_0)$, where

T is a trellis as in fig.1,

Σ is a nonempty finite set called the input alphabet

$\Gamma, \Sigma \subset \Gamma$ is a nonempty finite set called the operating alphabet

$\Gamma_0 \subset \Gamma$ is the set of accepting symbols

$\delta: \Gamma \times \Gamma \to \Gamma \times \Gamma$ is the transition function of A.

With respect to the structure of automaton we can consider the processors to be labelled - p_0 is the input/output processor, p_{i+1} (p_{i-1}) is the right (left) neighbour of the processor p_i, $i > 0$.

It can easily be shown that the restriction $\delta(a,b)=(c,c)$ $\forall a,b \in \Gamma$ changes nothing in the computational power of 2LS (instead of $\delta(a,b) = (s,r)$ we can assume another transition function δ^* such that $\delta^*(a,b) = ((s,r),(s,r))$ and $\delta^*((s_1,r_1),(s_2,r_2)) = \delta(r_1,s_2)$ hold). This enables us to denote by $O_t(i)$ the output of the processor p_i in the time unit t.

In what follows $\mathcal{L}(2LS)$ denotes the family of languages recognized by 2LS automata.

2.Results

Now we are going to give the main result of this paper. Firstly,we desribe the language for which we show not to be recognizable by 2LS automata.

Let us consider the set of words $W_n = \{w \in \{0,1\}^* / |w| = 2^n, n \in N\}$. It is quite natural to regard each word $\alpha \in W_n$ as a code of a Boolean function of n variables. Obviously there is one-to-one relation between Boolean functions of n variables and the words from W_n.That is why we can denote by f_w the Boolean function coded by $w \in W_n$. Now we are ready to describe the language of our interest:

$L = \{ wv \in \{0,1\}^* / |w| = 2^{|v|}, f_w(v) = 1 \}$.

The following observation is simple but important.

Observation Let 2LS automaton A works on the input word w, $w = a_1 \ldots a_m$. Then the outputs $O_t(i)$, $t+i > m+1$ have no influence on the output of automaton in the time unit m+1.

Now we are ready to prove the following result.

Theorem L does not belong to $\mathcal{L}(2LS)$.

Proof: We shal prove this assertion by contradiction. Assuming that there is a 2LS automaton $A = (T, \Sigma, \Gamma, \delta, \Gamma_0)$ that accepts the language L, we shall construct two different words which are both accepted by A while one of them is from L and the second one is not.

We shall denote by $C^A_\omega(m,t)$, resp. $C(m,t)$, if it causes no confusion, the m-tuple of processors p_i, $i \leq m$, of 2LS automaton A working on the input word ω in time unit t. We shall also use the notion of m-subconfiguration for this m-tuple. It is clear that the number of different m-subconfigurations is at most $|\Gamma|^m$. We can consider them to be ordered; $\pi(C(m,t))$ will denote the order of the m-subconfiguration C(m,t) in the ordered sequence of m-subconfigurations.

Now, let us partition the class W_n into the subclasse according to $C^A_\omega(n, 2^n+1)$ as follows:

$V_i = \{\omega \in W_n / \pi(C^A_\omega(n, 2^n+1)) = i\}$, i.e. $|\{V_i\}| \leq |\Gamma|^n$.

Since the number of different Boolean functions of n variables is 2^{2^n}, there has to be (for suitable n) an integer i such that V_i contains two different words w,u. Therefore the word $\alpha \in \{0,1\}^n$ for which $f_w(\alpha) \neq f_u(\alpha) = 1$ is true has to exist. (the different words from W_n code different Boolean functions of n variables).

Now let us consider two words $w\alpha$ and $u\alpha$; α ,w,u are those mentioned above. Then $C^A_{w\alpha}(n,2^n+1) = C^A_{u\alpha}(n,2^n+1)$ holds. By observation this implies that 2LS automaton A cannot distinguish $w\alpha$ and $u\alpha$ which contradicts the fact that $w\alpha \notin L$ and $u\alpha \in L$.

□

The 2LS automaton works in the shortest possible time. Since an automaton based on linear systolic array with two-way communication is as powerful as Turing machine it is quite natural to consider automata working in a longer time.

Let 2LS-f(m) denote a 2LS automaton working in time f(m), $f:N\to N$, m is the length of the input word, $f(m) \geq m$. Then the following corolary holds.

Corolary $L \notin \mathcal{L}(2LS-(\lceil m + m^a \rceil))$, $a \in \langle 0,1)$.

Proof: It is sufficient to realize that if a 2LS - f(m) automaton A recognizes the input word $w=w_1 \ldots w_m$, then the outputs $O_t(i)$, $i > b + f(m)$ have no influence on the output of automaton A in the time unit t+b. Now the proof can be done analogous to that of Theorem:

- the partioning of the class W_n into the subclasses must be done according to $C^A_\omega(n+\lceil (n+2^n)^a \rceil, 2^n+1)$ instead of $C(n,2^n+1)$;
- the existence of an integer i such that V_i contains two different words is guaranted by validity of

$$0 \leq \lim_{n\to\infty} \frac{|\{V_i\}|}{2^{2^n}} \leq \lim_{n\to\infty} \frac{2^{n+(n+2^n)^a}}{2^{2^n}} = 0, \quad a \in \langle 0,1) .$$

□

We note that the described technique can be also used in the cases when the i-th processor of linear array of pro-

cessors with two-way communication has constant number of "neighbours" i.e. there is a constant k such that p_i cannot comminicates with the procesor p_j, $j < i-k$ or $j > i+k$.

REFERENCES

[1] Brent,R.P.,Kung,H.T. (1984) :Systolic arrays for Polynomial GCD computation.IEE Trans.Computers,(6) C-33,731-736.

[2] Gruska,J.,Ružicka,P.,Wiedermann,J.(1983) :Systolické systémy.Zborník SOFSEM'83,267-307.

[3] Janetka,I. (1986) : Real time computation by homogeneous structures,Proc.IMYCS'86,165-171.

[4] Kung,H.T.(1979) :Let's design algorithms for VLSI systems. CMU Comp.Sci.department,Techn.report.

[5] Kung,H.T. (1981) :Use of VLSI in algebraic computation:some suggestions.Proc. SYMSAC'81. ACM Symp.,Snowbind,Utah,218-222.

[6] Kung,H.T.(1982) : Why systolic architectures?,Computer Mag. 15 1 ,Jan.,37-46

[7] Kung,H.T.,Leiserson,C.E.(1978) :Systolic array for VLSI . Technical report,Dept. of Comp.Sci.,Carnegie-Mellon Univ.

[8]Thompson,C.D.(1980) : A complexity theory for VLSI.Dissertation CMU,Computer Sci.Department.

IMYCS'88, Smolenice Castle, November 14-18, 1988.

A REMARK ON SOME CLASSIFICATIONS OF INDIAN PARALLEL LANGUAGES

Bernd REICHEL

Department of Mathematics
Technological University
PSF 124
Magdeburg
DDR - 3010
German Democratic Republic

A b s t r a c t. We discuss the complexity measures Var, Prod, and Symb for Indian parallel languages.

1. INTRODUCTION AND PRELIMINARIES

A good review about the theory of descriptional complexity of context-free languages is given in [6]. It is stated that the knowledge about descriptional complexity of other families of languages is very small in relation to the situation of context-free languages. For this reason in this paper we study some classifications of Indian parallel languages. Indian parallel languages (IPL) and Indian parallel grammars (IPG) were first investigated in [9] and [10]. It is assumed that the reader is familiar with basic concepts concerning formal language theory. We refer, for instance, to [7].

An Indian parallel grammar G is a 4-tuple (N, T, P, S) where N is the alphabet of nonterminals, T is the alphabet of terminals, $N \cap T = \emptyset$, $N \cup T = V$, P is the set of productions, and $S \in N$ is the initial nonterminal. A word w_1 over V generates a word w_2 ($w_1 \Rightarrow w_2$) if and only if

i) $w_1 = x_0Ax_1A\ldots Ax_n$ where $A \in N$, $x_i \in (V \setminus \{A\})^*$ for $i = 0,1,\ldots,n$,

ii) $w_2 = x_0wx_1w\ldots wx_n$, and

iii) $A \to w$ is in P.

The Indian parallel language generated by the IPG G is defined by $L(G) = \{w \in T^*: S \overset{*}{\Rightarrow} w\}$ where $\overset{*}{\Rightarrow}$ denotes the reflexive and transitive closure of the direct derivation $\Rightarrow$. If it is not stated otherwise, then grammar and language are standing for Indian parallel grammar and Indian parallel language, respectively.

A grammatical complexity measure K is a mapping from the set of IPG into the set of natural numbers. Let further $K(L) = \min \{K(G): G \text{ is an IPG, } L(G) = L\}$. In this paper we investigate some classifications with respect to the size of an IPG. For an IPG $G = (N, T, P, S)$ we denote by Var(G) the number of its nonterminals and by Prod(G) the number of its productions. Symb(G) is the number of symbols, i. e. $\text{Symb}(G) = \sum_{A \to w \in P} (|w| + 2)$.

We shall show that the defined measures are connected, i. e. for each natural number n, there is an IPL L such that $K(L) = n$. In the second part we shall show that, in general, for given $n \in \mathbb{N}$ and given IPG G, it is undecidable whether or not $K(L(G)) = n$. For results of analogous problems concerning context-free languages, see [2], [3], [4], [5]. Because of space limitation not all proofs are complete. A full version and further results will appear ([8]).

2. CONNECTIVITY

In this section, we shall show that the defined complexity measures are connected. First we prove a useful lemma.

Lemma 2.1. Let p be a prime number and $L_p = \{a^{p^i}: i \geq 0\}$. Then $\text{Symb}(L_p) = p + 5$.

Proof. Let $G_p = (N, T, P, S)$ be a reduced IPG generating L_p with $\text{Symb}(L_p) = \text{Symb}(G_p)$. Because L_p is a language over an one-letter-alphabet, we can renounce the order of letters in

a word $w \in V^*$. Therefore, without loss of generality we write $A_1 \overset{*}{\Rightarrow} a^{i_0} A_1^{i_1} A_2^{i_2} \ldots A_n^{i_n}$ for a derivation $A_1 \overset{*}{\Rightarrow} w$ where $\#_a(w) = i_0$ and $\#_{A_j}(w) = i_j$ for $j = 1,\ldots,n$.

Because L_p is not a CFL, there must be a nonterminal $A \in N$ which appear in following derivations $S \overset{*}{\Rightarrow} a^{k_0} A^{k_1}$ where $k_0 \geq 0$, $k_1 \geq 1$ and $A \overset{*}{\Rightarrow} a^{m_0} A^{m_1}$ where $m_0 \geq 0$, $m_1 \geq 1$ (if $m_0 = 0$, then $m_1 \geq 2$). Let further $\{a^n\} \subseteq G(A)$ for some $n \geq 1$. Then the following derivations are possible:

$$S \overset{*}{\Rightarrow} a^{k_0} A^{k_1} \overset{*}{\Rightarrow} a^{k_0}(a^n)^{k_1} = a^{p^{s_0}}$$
$$S \overset{*}{\Rightarrow} a^{k_0} A^{k_1} \overset{*}{\Rightarrow} a^{k_0}(a^{m_0} A^{m_1})^{k_1} \overset{*}{\Rightarrow} a^{k_0}(a^{m_0} a^{m_1 n})^{k_1} = a^{p^{s_1}}$$
$$\vdots \qquad\qquad \vdots \qquad\qquad \vdots$$
$$S \overset{*}{\Rightarrow} a^{k_0} A^{k_1} \overset{*}{\Rightarrow} a^{k_0}(a^{m_0} a^{m_0 m_1} \ldots a^{m_0 m_1^{i-1}} A^{m_1^i})^{k_1} \overset{*}{\Rightarrow}$$
$$\overset{*}{\Rightarrow} a^{k_0}(a^{m_0} a^{m_0 m_1} \ldots a^{m_0 m_1^{i-1}} a^{m_1^i n})^{k_1} = a^{p^{s_i}} \quad \text{where } s_i < s_{i+1}$$

for $i \geq 0$. If we consider the terminal words of above derivations, we get for s_j, $j \geq 0$ the following equalities:

$$k_0 + k_1 n = p^{s_0} \text{ and}$$
$$k_0 + k_1(m_0 + m_0 m_1 + \ldots + m_0 m_1^{i-1} + m_1^i n) = p^{s_i} \text{ for } i \geq 1.$$

From this $p^{s_{i+1}} - p^{s_i} = k_1 m_1^i (m_0 + m_1 n - n)$ and with $p^{s_1} - p^{s_0} = k_1(m_0 + m_1 n - n)$ finally

$$p^{s_{i+1}} - p^{s_i} = m_1^i (p^{s_1} - p^{s_0}) \qquad (1)$$

follows. Especially, for i=1 we get $p^{s_2} - p^{s_1} = m_1(p^{s_1} - p^{s_0})$. Because of $s_0 < s_1 < s_2$ then $p^{s_2 - s_0} - p^{s_1 - s_0} = m_1(p^{s_1 - s_0} - 1)$ and further $p^{s_2 - s_0} + m_1 = (m_1 + 1)p^{s_1 - s_0}$. Since $p^{s_1 - s_0}$ clearly devides $p^{s_2 - s_0}$, $p^{s_1 - s_0}$ must devide m_1, too. Therefore, $m_1 = p^m$ for some $m \geq 0$. From (1) then $p^{s_{i+1}} - p^{s_i} = p^{im}(p^{s_1} - p^{s_0})$ and further $p^{s_{i+1} - im} - p^{s_i - im} = p^{s_1} - p^{s_0}$. But then $p^{s_{i+1} - im} = p^{s_1}$ and $p^{s_i - im} = p^{s_0}$ must be and we finally get $p^{s_i} = (p^m)^i p^{s_0}$. Because of $s_i > s_0$ $m \geq 1$ must hold. We have shown now that for all derivations of G_p of the structure $A \overset{*}{\Rightarrow} a^{m_0} A^{m_1}$ $m_1 = p^m$ for some $m \geq 1$ must hold.

In the second part we shall show that for the derivation $A \overset{*}{\Rightarrow} a^{m_0} A^{p^m}$, $m \geq 1$, a production $B \to w$ where $|w| \geq p$ is necessary and can not be deleted. We consider the derivation

$$A = w_0 \Rightarrow w_1 \Rightarrow \ldots \Rightarrow w_n = a^{m_0} A^{p^m} \text{ where for all } i = 1,\ldots,n-1$$
$$w_i \neq a^{j_1} A^{p^{j_2}} \text{ for some } j_1 \geq 0,\ j_2 \geq 1 \text{ (otherwise we consider}$$

the derivation $A \overset{*}{\Rightarrow} w_i$). In order to get a contradiction, we assume that only rules $B \to w$ with $|w| < p$ are used for the above derivation. From this it follows that there is an i_0, $1 \leq i_0 \leq n-1$, such that in w_{i_0} at least two different nonterminals appear (otherwise we cannot generate A^{p^m} by Indian parallelism). Then we can consider two cases:

case 1. In w_{n-1} two different nonterminals appear, then one of these must be A. Let $w_{n-1} = a^{b_1}A^{b_2}B^{b_3}$ where $B \neq A$, $b_2 \geq 1$, $b_3 \geq 1$. Then we have $A \overset{*}{\Rightarrow} a^{b_1}A^{b_2}B^{b_3} \Rightarrow a^{m_0}A^{p^m}$ with $B \to a^{b_4}A^{b_5} \in P$. From this we get the condition

$$b_2 + b_3 b_5 = p^m \qquad (2).$$

Let us assume that no production $B \to w$, $w \in V^*$, is used in the above derivation up to w_{n-2} (otherwise we consider $B \overset{*}{\Rightarrow} a^{m'_0}B^{p^{m'}}$ for some $m' \geq 1$). Then we can construct the derivation

$A \overset{*}{\Rightarrow} a^{b_1}A^{b_2}B^{b_3} \overset{*}{\Rightarrow} a^{b_1}(a^{b_1}A^{b_2}B^{b_3})^{b_2}B^{b_3} \overset{*}{\Rightarrow}$

$\overset{*}{\Rightarrow} a^{b_1+b_1b_2+b_4(b_2b_3+b_3)}A^{b_2b_2+b_5(b_2b_3+b_3)}$. Then the following equality for some $m' > 1$ must hold: $b_2b_2+b_5b_3+b_5b_2b_3 = p^{m'}$ or $b_2(b_2+b_3b_5)+b_3b_5 = p^{m'}$. With (2) we get $b_2p^m+b_3b_5 = p^{m'}$. Because of $m' > m$ p^m must devide b_3b_5. If $m'' \geq 1$ and $p^{m''}$ devides b_5, then we have $B \to a^{b_4}A^{g_1p^{m''}} \in P$ for some $g_1 \geq 1$, a contradiction. Therefore, we have to assume $b_5 = 1$. Hence, $b_3 = g_2p^m$ for some $g_2 \geq 1$. But then from (2) it follows that $g_2 = 1$ and $b_2 = 0$, a contradiction.

case 2. In w_{n-1} only one nonterminal appears, say B. Obviously, $B \neq A$. Let $w_{n-1} = a^{c_1}B^{c_2}$. Then we have $A \overset{*}{\Rightarrow} a^{c_1}B^{c_2} \Rightarrow a^{m_0}A^{p^m}$. From this follows $c_2 = p^m$ and $B \to a^{c_3}A \in P$. But then we can consider the derivation $B \Rightarrow a^{c_3}A \overset{*}{\Rightarrow} a^{c_3+c_1}B^{p^m}$.

We have proved now that, if $G_p = (N, T, P, S)$ generates L_p, then there is at least one rule $B \to w$ with $|w| \geq p$ in P. Therefore, $G_p = (\{S\}, \{a\}, \{S \to S^p, S \to a\}, S)$ is minimal for L_p with respect to Symb. Hence, $\mathrm{Symb}(L_p) = p + 5$.

Remark. From the lemma it follows that there are IPL, which have no IPG in CHOMSKY - Normalform.

Theorem 2.2. For every integer $n \geq 1$, there exists a IPL L_n such that $Var(L_n) = n$.

Proof. The theorem trivially holds for $n = 1$. For $n > 1$ one can easily show by using the proof of lemma 2.1. that for $L_n = \{ba^{p_1^{i_1}} ba^{p_2^{i_2}} b \ldots ba^{p_{n-1}^{i_{n-1}}} b$: for $j = 1,\ldots,n-1$ p_j are different prime numbers, $i_j \geq 0\}$ $Var(L_n) = n$ holds.

Theorem 2.3. For every integer $n \geq 1$, there exists a IPL L_n such that $Prod(L_n) = n$.

Proof. For $n = 1$ the theorem trivially holds. For $n > 1$ we use the IPL $L_n = \{aba^2b \ldots a^ib : 1 \leq i \leq n\}$ to show the theorem. Obviously, $Prod(L_n) \leq n$ holds. Let $G_n = (N, \{a,b\}, P, S)$ be a reduced IPG such that $L(G_n) = L_n$ and $Prod(G_n) = Prod(L_n)$. If $A \neq S$ is a nonterminal in G_n, then $|G(A)| \geq 2$. Otherwise, there would exist a grammar G_n' for L_n having fewer productions than G_n. It is easy to see that there exists no derivation $A \overset{*}{\Rightarrow} xAy$ where $xy \in V^+$.

First one can prove by quite technical and numerical investigations that there exists no nonterminal $A \neq S$ such that there is a derivation $S \overset{*}{\Rightarrow} x$ such that $\#_A(x) \geq 2$.

In the second part, by similar considerations, one can show that the grammar G_n must be linear, i. e. there are no nonterminals A and B in N such that $S \overset{*}{\Rightarrow} xAyBz$ where $xyz \in T^*$.

Finally, in the last part, by similar quite technical investigations can be shown that G_n contains only productions of the form $S \rightarrow w$, $w \in T^*$, which would complete the proof of the theorem.

Theorem 2.4. For every integer $n \geq 2$, there is an IPL L_n such that $Symb(L_n) = n$.

Proof. The proof of theorem 1 in [5] for the complexity measure Symb with respect to context-free languages can be used in the same manner for the proof of this theorem.

3. DECISION PROBLEMS

In [6] are formulated some decision problems concerning complexity measures. One of the basic questions is the following one:

Is it decidable, for given integer n and IPG G, whether or not $K(L(G)) = n$?

In this section, we shall discuss this question for the defined complexity measures Var, Prod, and Symb of Indian parallel languages. We give results without proofs. The proofs are realized by using the POST Correspondence Problem. They are quite technical and they are based on results of the second paragraph and constructions and ideas which are used for corresponding problems in [1], [4], and [5].

Theorem 3.1. For every integer $n \geq 1$, it is undecidable, for an arbitrary IPG G, whether or not $Var(L(G)) = n$.

Theorem 3.2. It is undecidable, for given $n \geq 3$ and for an arbitrary IPG G, whether or not $Prod(L(G)) = n$.

Remark. Obviously, for $n = 1$ it is decidable, for an arbitrary IPG G, whether or not $Prod(L(G)) = 1$. But for $n = 2$ the problem is open.

Theorem 3.3. It is decidable (undecidable), for given integer $n \leq 5$ $(n \geq 10)$ and for an arbitrary IPG G, whether or not $Symb(L(G)) = n$.

Remark. For $6 \leq n \leq 9$, it is an open problem whether it is decidable or not, for given integer n and arbitrary IPG G, whether or not $Symb(L(G)) = n$.

Acknowledgement. I wish to thank Dr. J. Dassow for many useful discussions.

REFERENCES

[1] S. Ginsburg, The Mathematical Theory of Context-Free Languages (McGraw-Hill, New York, 1966).

[2] J. Gruska, On a classification of context-free languages, Kybernetika 3 (1967) 22-29.

[3] J. Gruska, Some classifications of context-free languages, Information and Control 14 (1969) 152-179.

[4] J. Gruska, Complexity and unambiguity of context-free grammars and languages, Information and Control 18 (1971) 502-512.

[5] J. Gruska, On the size of context-free grammars, Kybernetika 8 (1972) 213-218.

[6] J. Gruska, Descriptional complexity of context-free languages, in: Proc. MFCS'73 High Tatras (1971) 71-83.

[7] M.A. Harrison, Introduction to Formal Language Theory (Addison-Wesley P. C., Reading, Massachusetts, 1978)

[8] B. Reichel, Some classifications of Indian parallel languages, to appear.

[9] R. Siromoney and K. Krithivasan, Parallel context-free languages, Information and Control 24 (1974) 155-162.

[10] S. Skyum, Parallel context-free languages, Information and Control 26 (1974) 280-285.

IMYCS'88, Smolenice Castle, November 14-18, 1988.

SIX ARITHMETIC-LIKE OPERATIONS ON LANGUAGES

LILA SANTEAN

Institute for Computers and Informatics
Department of Knowledge Processing Systems
8-10 Miciurin blvd.
71316 Bucharest 1
Romania

Operations on languages are intensivelly studied in formal language theory. For example, there are representations of some families of languages starting from simpler languages and using suitable operations, finding of counter-examples often uses operations on languages, the theory of abstract families of languages (AFL) studies just operations, many operations appear in formal language theory applications [1], and so on.

The existing operations can be roughly clustered in three classes: set operations (union, intersection, complementation), algebraic operations (homomorphism, substitution) and purely language theoretical operations (Kleene closure, shuffle). Within this frame, it is obvious to ask for language operations corresponding to the arithmetic operations on numbers: sum, product, power, factorial, square root, and so on. Six such operations will be defined and investigated in the following, namely the compact subtraction, the literal subtraction, the generalized subtraction, the multiplication, the power, and the factorial.

The aim of this paper is to examine the closure of an abstract family of languages (when positive results are true) or directly of families in Chomsky hierarchy (when negative results hold) under these operations.

Generally, the results are the expected ones, in the sense that the family of context-sensitive languages is not closed under erasing operations, whereas for the families of context-free and regular languages, the situation is

---

On a leave from the Regional Computing Center, Tulcea

just the opposite.

1. COMPACT SUBTRACTION

For a vocabulary V, we denote by V^* the free monoid generated by V under the concatenation operation; the null element of V is λ and $|x|$ denotes the length of the string $x \in V^*$. The four families in Chomsky hierarchy are denoted $\mathcal{L}_i$, i = 0, 1, 2, 3 ($\mathcal{L}_{lin}$ denotes the family of linear languages). For other notations and terminologies in formal language theory, the reader is referred to [2].

DEFINITION 1.1 Let L_1, L_2 be languages on V^*. We define the compact subtraction of L_1 and L_2 by:

$$L_1 \ominus L_2 = \bigcup_{\substack{x \in L_1 \\ y \in L_2}} (x \ominus y), \text{ where } x \ominus y = \left\{ z \in V^* / z = x_1 x_2, x = x_1 y x_2 \right\}.$$

Compact subtraction is a generalization of right or left quotient: instead of extracting the word y from the left or right extremity of x, we extract it from an arbitrary place in x.

THEOREM 1.1 $\mathcal{L}_1$ is not closed under compact subtraction.

Proof: If L_1, L_2 are two languages on V^*, we notice that $\{c\} L_1 \ominus \{c\} L_2 = L_2 \setminus L_1$, where c is a symbol which doesn't belong to V. As the family $\mathcal{L}_1$ is not closed under left quotient with regular languages, it follows that it is not closed under operation $\ominus$, either.

THEOREM 1.2 If L_1 and L_2 are languages on V^*, L_2 a regular one, there is a gsm g (with erasing) so that $L_1 \ominus L_2 = g(L_1)$.

Proof: Let $A = (K, V, s_0, F, P)$ be a finite automaton that recognizes L_2. We construct the gsm:

$g = (V, V, K \cup \{s_0', s_f\}, s_0', \{s_f\}, P')$, where

$$P' = \{s_0'a \to as_0' / a \in V\} \cup P \cup \{s_0'a \to s / s_0a \to s \in P\}$$
$$\cup \{sa \to s_f / sa \to s' \in P,\ s' \in F\} \cup \{s_f a \to as_f / a \in V\}$$
$$\cup \{s_0'a \to s_f / s_0a \to s \in P,\ s \in F\} \cup \{s_0'a \to as_f / a \in V,\ \lambda \in L_2\}.$$

Clearly, $g(L_1) = L_1 \ominus L_2$ and thus proof is finished.

COROLLARY: $\mathcal{L}_2$, $\mathcal{L}_{lin}$, $\mathcal{L}_3$ are closed under compact subtraction with regular languages.

OPEN PROBLEMS: The closure of the families $\mathcal{L}_2$ and $\mathcal{L}_{lin}$ under compact subtraction.

Probably, these families are not closed under

compact subtraction, or, if they are, this result cannot be proved in a constructive way, because we have:

THEOREM 1.3 There is no algorithm to decide whether $L_1 \ominus L_2$ is empty or not, for L_1, L_2 arbitrary in $\mathcal{L}_{lin}$.

Proof: Let us consider the linear languages

$L_1 = \{ ada^{i_k}b \ldots ba^{i_1}bc\, x_{i_1} \ldots x_{i_k} d / k \geq 1,\ i_1, \ldots, i_k \in \{1, \ldots, n\} \}$,

$L_2 = \{ da^{i_k}b \ldots ba^{i_1}bc\, y_{i_1} \ldots y_{i_k} d / k \geq 1,\ i_1, \ldots, i_k \in \{1, \ldots, n\} \}$.

The statement: $L_1 \ominus L_2 = \emptyset$ iff there is a sequence of indexes $i_1, \ldots, i_k \in \{1, \ldots, n\}$ so that $x_{i_1} \ldots x_{i_k} = y_{i_1} \ldots y_{i_k}$, is obvious. Therefor, we have $L_1 \ominus L_2 \neq \emptyset$ iff the POST correspondence problem has a solution, which is undecidable.

Concluding, we cannot construct in an algorithmic way a context-free grammar G so that $L(G) = L_1 \ominus L_2$, $L_1, L_2 \in \mathcal{L}_{lin}$, as, otherwise we can decide if $L_1 \ominus L_2 = \emptyset$ (the problem if L(G) is empty, finite or infinite is decidable for context-free grammars) - contradiction.

2. LITERAL SUBTRACTION

DEFINITION 2.1 Let L_1, L_2 be languages on V^*. We define the literal subtraction, $L_1 -. L_2$, by $L_1 -. L_2 = \bigcup_{\substack{x \in L_1 \\ y \in L_2}} (x -. y)$, where

$x -. y = \{ x_1 x_2 \ldots x_k / x_1 b_1 \ldots b_{k-1} x_k = x,\ b_1 \ldots b_{k-1} = y,\ k \geq 2,\ b_i \in V,\ \forall i \in \{1, \ldots, k-1\},\ x_j \in V^*,\ \forall j \in \{1, \ldots, k\} \}$ (if the letters of y can also be found in x, in the same order, then the literal subtraction erases them from x, without taking into account their places; else we cannot subtract y from x).

THEOREM 2.1 If L_2 is a regular language, then the literal subtraction $L_1 -. L_2$ can be attained by a gsm (with erasing).

Proof: Let $A = (K, V, s_0, F, P)$ be a finite automaton that recognizes the language L_2 (therefor P contains rules of the form $sa \rightarrow s'$, $s, s' \in K$, $a \in V$).

We construct the gsm $g = (V, V, K, s_0, F, P')$ with K, V, s_0, F according to A and $P' = P \cup \{ sa \rightarrow as / s \in K,\ a \in V \}$. One can easily prove that $L_1 -. L_2 = g(L_1)$ (the rules of P erase the symbols which come from y, in the correct order, and those of the form $sa \rightarrow as$ cross the symbols that will

remain in x-.y).

COROLLARY: $\mathcal{L}_3, \mathcal{L}_{lin}, \mathcal{L}_2$ are closed under literal subtraction with regular languages.

THEOREM 2.2 $\mathcal{L}_1$ is not closed under literal subtraction with regular languages.

Proof: We define the gsm $g = (V, V \cup V', K, s_0, F, P')$, where $K = \{s_0, s\}$, $F = \{s\}$, $V' = \{a'/a \in V\}$, $P' = \{s_0 a \to a s_0 / a \in V\} \cup \{s_0 a \to a' s / a \in V\} \cup \{sa \to a's / a \in V\}$. If $L \subseteq V^*$, we have the relation: $g(L) = \{w_1 w_2' / w_1 w_2 \in L\}$ (the gsm g marks the symbols that are situated on the right side of the symbols of L). We also have the relation: $L_1/L_2 = [g(L_1) -. h(L_2)] \cap V^*$, where $L_1, L_2 \subseteq V^*$ and h is a homomorphism, $h: V \to V'$, $h(a) = a'$.

As $\mathcal{L}_1$ is closed under intersection but it is not closed under right (and left) quotient with regular languages, it follows that $\mathcal{L}_1$ is not closed under operation -.

THEOREM 2.3 $\mathcal{L}_2$ and $\mathcal{L}_{lin}$ are not closed under literal subtraction with linear languages.

Proof: Let L_1, L_2 be the linear languages $L_1 = \{a^n (bc)^n (df)^m / n, m \geqslant 1\}$, $L_2 = \{c^n d^n / n \geqslant 1\}$. One can easily see that:

$$[L_1 -. L_2] \cap \{a\}^* \{b\}^* \{f\}^* = \{a^n b^n f^n / n \geqslant 1\}.$$

As $\mathcal{L}_1$ and $\mathcal{L}_{lin}$ are closed under intersection by regular sets but $\{a^n b^n f^n / n \geqslant 1\}$ is not a context-free language, it follows that these families are not closed under literal subtraction. In fact, we have obtained a stronger result, namely that there are linear languages L_1, L_2 such that $L_1 -. L_2$ is not a context-free language.

3. GENERALIZED SUBTRACTION

DEFINITION 3.1 Let L_1, L_2 be languages on V^*. We define the generalized subtraction $L_1 \not- L_2$ by:

$$L_1 \not- L_2 = \bigcup_{\substack{x \in L_1 \\ y \in L_2}} (x \not- y), \text{ where } x \not- y = \{x_1 x_2 \ldots x_{k+1} /$$

$x = x_1 b_1 x_2 b_2 \ldots x_k b_k x_{k+1}$, where y is a permutation of the word $b_1 b_2 \ldots b_k$, $k \geqslant 1\}$ (if the letters of y can also be found in x, then the generalized subtraction erases the letters of y from x without taking into account their places; else we cannot subtract y from x). Notice that the generali-

zed subtraction is a generalization of the compact and literal subtraction.

THEOREM 3.1 $\mathcal{L}_3$ is not closed under generalized subtraction.

Proof: Let L_1, L_2 be the regular languages

$L_1 = \{ (bc)^m (df)^p / m, p \geq 1 \}$, $L_2 = \{ (cd)^n / n \geq 0 \}$.

One can prove that

$(L_1 \not- L_2) \cap \{b\}^* \{f\}^* = \{ b^m f^m / m \geq 1 \}$.

As $\mathcal{L}_3$ is closed under intersection by regular languages but $\{ b^m f^m / m \geq 1 \}$ is not regular, it follows that $\mathcal{L}_3$ is not closed under operation $\not-$.

THEOREM 3.2 $\mathcal{L}_{lin}$, $\mathcal{L}_2$ are not closed under generalized subtraction with regular languages.

Proof: Let L_1, L_2 be the linear languages:

$L_1 = \{ a^n (bc)^n (df)^m / n, m \geq 1 \}$, $L_2 = \{ (cd)^n / n \geq 1 \}$.

$(L_1 \not- L_2) \cap \{a\}^* \{b\}^* \{f\}^* = \{ a^n b^n c^n / n \geq 1 \}$ is an obvious relation.

As $\mathcal{L}_2$ and $\mathcal{L}_{lin}$ are closed under intersection by regular languages but $\{ a^n b^n c^n / n \geq 1 \}$ is not context-free, it follows that $\mathcal{L}_{lin}$ and $\mathcal{L}_2$ are not closed under generalized subtraction with regular languages.

THEOREM 3.3 $\mathcal{L}_1$ is not closed under generalized subtraction with regular sets.

Proof: For each $L_o \in \mathcal{L}_o$ (hence also for $L_o \in \mathcal{L}_o - \mathcal{L}_1$, $L_o \subseteq V^*$), there is $L_1 \in \mathcal{L}_1$, $L_1 \subseteq \{a\}^* b L_o^*$, $a, b \notin V$, such that for each $x \in L_o$ there is a natural p such that $a^n bx \in L_1$ ([2]). Consider such a language $L_1 \in \mathcal{L}_1$. We have $L_o = (L_1 \not- \{a\}^* b) \cap V^*$. As $\mathcal{L}_1$ is closed under intersection, it follows that it cannot be closed under generalized subtraction with regular sets.

4. MULTIPLICATION

DEFINITION 4.1 Let L_1, L_2 be languages on V^*. We define their multiplication by:

$L_1 * L_2 = \{ x^{|y|} / x \in L_1, y \in L_2 \}$ on condition that $\lambda^{|\alpha|} = \lambda$, $\forall \alpha \in L_2$ and $\beta^{|\lambda|} = \lambda$, $\forall \beta \in L_1$.

THEOREM 4.1 $\mathcal{L}_3$ is not closed under multiplication.

Proof: Let L_1, L_2 be the regular languages

$L_1 = \{ a^n b / n \geq 1 \}$, $L_2 = \{ aaa \}$. In accordance with the definition 4.1 we have $L_1 * L_2 = \{ a^n b a^n b a^n b / n \geq 1 \}$, which is not even context-free.

COROLLARY The families $\mathcal{L}_2$ and $\mathcal{L}_{lin}$ are not closed under multiplication.

THEOREM 4.2 $\mathcal{L}_1$ is closed under multiplication.

Proof: For the detailed proof see [3].

5. POWER

DEFINITION 5.1 If L_1 and L_2 are languages on V^*, we define $L_1 ** L_2$ (L_1 power L_2) by:

$L_1 ** L_2 = \{ x_1 |x_2| \cdot |x_3|^2 \cdot \ldots \cdot |x_{|z|}|^z / x_i \in L_1, 1 \leq i \leq z, z \in L_2 \}$ on condition that if $\lambda \in L_1$ or $\lambda \in L_2$, we put λ in $L_1 ** L_2$.

THEOREM 5.1 $\mathcal{L}_3$ is not closed under operation $**$.

Proof: Let L_1, L_2 be the regular languages:

$L_1 = \{ aa \}$, $L_2 = \{ a^n / n \geq 1 \}$.

Then, $L_1 ** L_2 = \{ a^{2^n} / n \geq 1 \}$, language that is not even context-free.

COROLLARY: $\mathcal{L}_2$, $\mathcal{L}_{lin}$ are not closed under operation $**$.

THEOREM 5.2 If $L_1 \in V^2 V^*$, $L_2 \subseteq V^*$, $L_1, L_2 \in \mathcal{L}_1$, then $L_1 ** L_2 \in \mathcal{L}_1$.

Proof: We omit the proof which is detailed in [3].

OPEN PROBLEM: Is $\mathcal{L}_1$ closed under operation $**$?

6. FACTORIAL

DEFINITION 6.1 Let L be a language on V^*. We define L factorial by: $L! = \{ x! / x \in L \}$ where, if $x = a_1 a_2 \ldots a_n$, then $x! = a_1 \; a_1 a_2 \; a_1 a_2 a_3 \ldots a_1 a_2 a_3 \ldots a_n$, on condition that $\lambda ! = \lambda$ and $a! = a$, $\forall a \in V$.

THEOREM 6.1 $\mathcal{L}_3$ is not closed under operation !.

Proof: Let L be the regular language $L = \{ a^n / n \geq 1 \}$.

In accordance to definition 6.1, $L ! = \{ a^{n(n+1)/2} / n \geq 1 \}$, language which is not even context-free.

COROLLARY: $\mathcal{L}_2$, $\mathcal{L}_{lin}$ are not closed under operation !.

THEOREM 6.2 $\mathcal{L}_1$ is closed under operation !.

Proof: For the details of the proof the reader is referred to [3].

ACKNOWLEDGEMENT

We wish to express our gratitude to dr.GHEORGHE PAUN for his suggestions and remarks.

REFERENCES

[1] GH.PAUN, Generative Mechanisms for Economic Processes - in Romanian - , (Editura Tehnică, Bucureşti, 198o).

[2] A. SALOMAA, Formal Languages (Academic Press, New York, London, 1973).

[3] L. SANTEAN, Formal Languages Problems Suggested by Number Theory - in Romanian -, Grad. Disertation, Faculty of Mathematics, University of Bucharest, 1987.

IMYCS'88, Smolenice Castle, November 14-18, 1988.

SOME PROPERTIES OF SPACE-BOUNDED SYNCHRONIZED ALTERNATING TURING MACHINES WITH ONLY UNIVERSAL STATES

ANNA SLOBODOVÁ
Department of Theoretical Cybernetics, Comenius University
842 15 Bratislava, Czechoslovakia

1. PRELIMINARIES

Recently many parallel models have been investigated. One of them - alternation - was introduced in |1| as a generalization of nondeterminism. In related papers |4 - 16| the investigations continued. In |16| a synchronization as a form of communication among parallel processes in alternating computations was introduced. Here we restrict ourselves on a space-bounded synchronized alternating Turing machines with only universal states.

First we give necessary terminology and notations. The formal definitions can be found in |16|. The next section contains our results.

A synchronized alternating Turing machine (satm) has a read-only input tape with left and right endmarkers, and one semi-infinite read-write storage tape with a left endmarker. Input head and storage tape head can move to the right and also to the left. The set of states is partitioned into accepting,

rejecting, existential and universal states. satm can be considered as an alternating Turing machine - atm (defined in |1|) some states of which (sync states) have a sync element from a given finite set. The communication is through these elements mediated. When a process enters a sync state, it stops and waits until all parallel processes either enter the states with the same sync element or stop in final states.

A step of satm consists of reading one symbol from each tape, writing one symbol on the storage tape, moving the input and the storage head in specified directions, and entering a new state from a final control, in accordance with the next move relation.

A configuration of satm is given by the input, the input head position, the current state, the contents of the storage tape, and the storage tape head position. The last three components present an internal configuration. The initial configuration of satm M on input x is $I_M(x) = (x,1,q_o, \varepsilon ,1)$, where q_o is initial state of M and ε is the empty string. A configuration is called existential,universal, accepting, or rejecting, resp., if the state associated with it is existential, universal, accepting, or rejecting, respectively. Similarly we define sync and non-sync configurations.

Given satm M we write $\beta \vdash_M \beta'$ and say β' is a succesor of β if the configuration β' follows from the β in one step of M. $\vdash_M^*$ denotes the reflexive and transitive closure of relation $\vdash_M$.

A computational path of M is a sequence of configurations $\beta_o \vdash_M \beta_1 \vdash_M \ldots \vdash_M \beta_m$, for any $m \geq 0$. If $\beta_o = I_M(x)$, for some x, we call this sequence the sequential computation of M on x.

Let C be a sequential computation of M and $\beta_1 \vdash_M^* \ldots \ldots \vdash_M^* \beta_r$ be a subsequence of C that consists of all sync configurations of C. Suppose, for all j, such that $1 \leq j \leq r$, S_j is the sync element of β_j. Then $S_1,\ldots,S_r$ is called the sync sequence of C.

A computation of M is a finite nonempty labelled tree V with the following properties:

1. Each node u of V is labelled with a configuration l(u).

2. If u is an internal (non-leaf) node of V and $l(u)$ is existential, then u has exactly one child v such that $l(u)|\overline{\overline{M}}l(v)$.
3. If u is an internal node of V, $l(u)$ is universal and $\{\beta / l(u)|\overline{\overline{M}}\beta\} = \{\beta_1,\ldots,\beta_n\}$, then u has exactly n children $v_1,\ldots,v_n$ such that $l(v_i) = \beta_i$, for all i; $1 \leq i \leq n$.
4. For any two sync sequences $S^1 = S^1_1,\ldots,S^1_p$; $S^2 = S^2_1,\ldots,S^2_r$, corresponding with two paths, beginning in the root of the tree V, $S^1_i = S^2_i$ for all i; $1 \leq i < \min\{p,r\}$ must be true.

An accepting computation of M on x is a computation of M the root of which is labelled with $I_M(x)$, the leaves are labelled with accepting configurations and 4) is true for all i; $1 \leq i \leq \min\{p,r\}$.

The longest sync sequence over all the sequential computations of the accepting computation V is called a sync sequence of V.

We define the language recognized by the satm M as the set $L(M) = \{x \mid$ there is an accepting computation of M on $x\}$, in natural way. We say that two satm M and N are equivalent if $L(M)=L(N)$.

k-tape synchronized alternating Turing machine ($satm_k$) can be defined similarly.

We say a satm is a one-way satm (and write 1 satm) if it can move its input head only to the right.

The (synchronized) alternating Turing machine which has no existential states is denoted (s)utm and is called (synchronized) alternating Turing machine with only universal states.

The computational complexity of satm is defined as in |16|. It is mentioned as the greatest complexity over all accepting computations of the machine on all inputs with the same length Suppose M is a satm and V is an accepting computation of M.

For any configuration ß let space (ß) denote the sum of the length of the nonblank storage tapes contents in ß. Then the space of V is $\varphi(V) = \max\{\text{space}(\beta)/\beta \text{ occurs in } V\}$.

Let $S_1,\ldots,S_m$ be a sync sequence of V and let time (S_i) denote the maximum number of steps of M from the (i-1)-th sync configuration (from the initial configuration if i=1) either to the i-th sync configuration or to an accepting one, if no other

sync configuration appears. The maximum is brought over all the computational paths in V. From the above it follows that time (S_i) is generally time between (i-1)-th and i-th synchronization. The time of V is $\mathcal{T}(V)=\sum_{i=1}^{m}$ time (S_i).

The length of the sync sequence of V - Syn(V) is called the synchronization of V.

We say the function $S_M(n):N\longrightarrow N$ is a space complexity of M if for all positive integers n holds $S_M(n)=\max\{\mathcal{S}(W)/W$ is an accepting computation of M on $x\in L(M)$ and $|x|=n$, where $|x|$ denotes the length of $x\}\cup\{1\}$. The time- and sync complexity of M can be defined in the same way (denoted by $T_M(n)$ and $Syn_M(n)$).

We say that satm M is f(n)-space bounded and denote satm (f(n))if $\forall_{n\in N}: S_M(n)\leq f(n)$ holds.
The class of languages recognized by some kind of devices is denoted by the same but capital letters as the respective device.

A function S(n) is space-constructible if there is a S(n)-space bounded deterministic Turing machine which for any input of the length n marks off exactly S(n) cells on the storage tape and stops.

2. RESULTS

In |16| was shown that any satm M can be simulated by an equivalent atm N such that $T_N(n)\leq 2T_M(n)$ and $S_N(n)=\max\{Syn_M(n),S_M(n)\}$. The proof of this result uses nondeterminism, therefore the same technique can not be applied on utm. The similar assertion concerning sutm and utm is nevertheless true.

Theorem 2.1: For any sutm M there is an utm_2 N such that L(M)=L(N). Moreover, for the computational complexity of N holds:

$T_N(n)\leq 3\,T_M(n)$

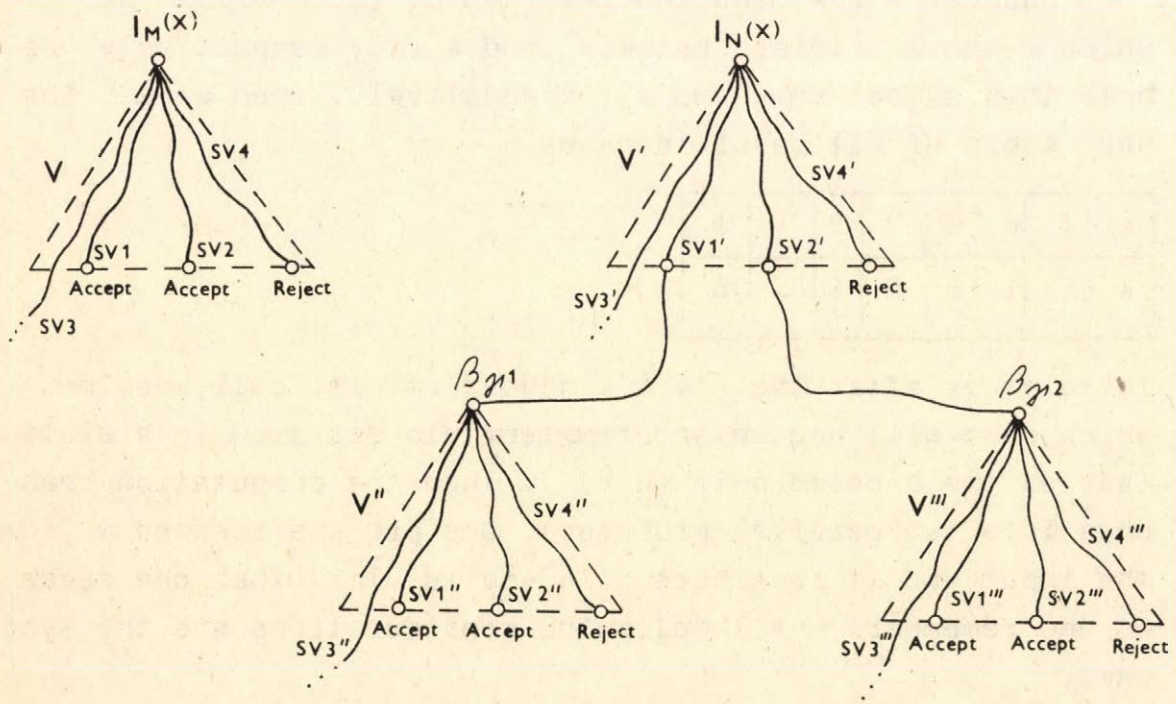
$S_N(n)=\max\{Syn_M(n),S_M(n)\}$

Outline of the proof: Let M be some sutm. An equivalent utm_2 N has one tape more, which shall be by the sync sequence of a simulated computation employed. N runs through the simulated computation two times.

During the first running it writes down the sync elements of the simulated sync configurations on its auxiliary storage tape in order they come. (This phase requires $T_M(n)$ time and $S_M(n) + Syn_M(n)$ space). During the simulation of an accepting configuration, N returns all its heads at the beginning of the tapes (max $\{S_M(n), Syn_M(n)\}$ time is needed for it). Then it runs over the simulated computation once more.

Several simulating computations run parallely now. Each of them has a sync sequence of one sequential computation on its storage tape, and vice-versa the sync sequence of each sequential computation of M which terminate in an accepting configuration is stored on the auxiliary storage tape in one of these parallel computations. N compares this sequence with the elements of the simulated sync configurations. They must agree. Clearly this phase of computation uses the same time and space as the first one.

The next picture can make the above-mentioned simulation more clear.



In |7,8,10| the alternating Turing machines with only universal states were investigated. Using the similar techniques as the ones employed in |10| we proved some assertions considering sutm with small space bound.

Lemma 2.2: Let $L_1 = \{w2w' / w,w' \in \{0,1\}^+, w \neq w'\}$.

Then 1) $L_1 \in$ 1 SATM(1)

2) $L_1 \in$ UTM(log n)

3) $L_1 \notin$ 1 SUTM (S(n)) for any $S(n) : N \longrightarrow R$ such that $\lim_{n \longrightarrow \infty} (S(n)) / n) = 0$

Proof:

1. We shall describe the computation 1 satm M which recognizes L_1 and does not use its storage tape. It can be determined deterministically whether the input has a form w2w', where $w,w' \in \{0,1\}^+$. It can be done during the one sequential computation. In the same time the computation, that assumes the input in the above-mentioned form runs parallely. It must check whether $w \neq w'$.

   Suppose $w \neq w'$ and the i-th symbol is the first one in which w and w' differ. Let w(i) and w'(i), respectively, denote this symbol in w and w', respectively. Then we get the next table of all possible cases

w(i)	0	1	2	2	0	1
w'(i)	1	0	0	1	∅	∅

   Immediately after the start M nondeterministically decides which case will happen and remembers this decision in a state (say as the ordered pair (u,v)). Then the computation branches into two parallel processes. One process scannes w from the input and it remembers u in states. The other one reads w and remembers v. All following configurations are the sync ones.

   At first the both processes set the sync element, in accordance to the symbol that is scanned. In certain moment M

nondeterministically decides, that the symbols scanned by these processes are distinct. In both processes M compares the scanned symbol with the one stored in its state. If these symbols agree M sets the special sync element. Clearly, the synchronization can be successful only if both decisions were right.

If $w = w'$ then such i that $w(i) \neq w'(i)$ does not exist. Hence the processes cannot set the special sync element and therefore they cannot accept the input. In opposed to it M rejects input after that the processes have read w and w', respectively.

2. Two-way alternating Turing machine N with only universal states such that $L(N) = L_1$ acts as follows.

Like in 1. it can be checked deterministically if the input has a form $w2w'$, where $w, w' \in \{0,1\}^+$. We parallely check if $w \neq w'$.

First N writes down 1 on the storage tape. In every moment this tape will store (in binary form) the positions of the compared symbols in w and w'.

N starts its action by the replacing its input head on the left-endmarker. If the storage tape stores i then N compares $w(i)$ and $w'(i)$. If $w(i) \neq w'(i)$ (besides the case $w(i) = 2$ and $w'(i) = \$$, when N rejects the input) then N accepts the input. If $w(i) = w'(i)$ then N increases the contents of the storage tape by one and repeats its action (like it was described above).

It is easy to see that N recognizes exactly L_1. Moreover N needs to store only the head position. Therefore it requires $\lceil \log \frac{n}{2} \rceil$ space.

3. Let O be a one-way synchronized alternating Turing machine with only universal states such that $L(O) = L_1$. Suppose O is S(n)-space bounded, and S(n) is such a function that $\lim_{n \to \infty}(S(n)/n) = 0$. Let r and s be the numbers of states and the storage tape alphabet of O, respectively. For each $n \geq 1$ let

$$V(n) = \{ w\,2\,w \mid w \in \{0,1\}^n \}.$$

For any $x \in V(n)$ we define the following sets of internal configurations:

$S(x) = \{ (q,\alpha,j)$ / there is a sequential computation of O on input x such that $I_0(x) \vdash^{+}_{O} (x,n+1,q',\alpha',j') \vdash_{O} (x,n+2,q,\alpha,j)$

Clearly for any $\sigma \in S(x)$ the configuration $(x,n+2,\upsilon)$ is the first one in which the input head is placed just behind the symbol 2.

$C(x) = \{ \{\sigma_1, \sigma_2\}$ / $\sigma_1 \in S(x)$, $\sigma_2 \in S(x)$ and the following assertions hold:

1. In a case $\sigma_1 = \sigma_2$ there is a computational path of O which starts in the configuration $(x,n+2,\sigma_1)$ and either terminates in a rejecting configuration or it is infinite.
2. In a case $\sigma_1 \neq \sigma_2$ there are two computational paths of O which start in the configurations $(x,n+2,\sigma_1)$ and $(x,n+2,\sigma_2)$, respectively, and they terminate in sync configurations with distinct sync elements.

Note that each $x \in V(n)$ is not accepted by O. Hence $C(x) \neq \emptyset$ for any $x \in V(n)$.

The next proposition help us to finish the proof.

Proposition 2.3: For any two different strings x and y in V(n)

$$C(x) \cap C(y) = \emptyset.$$

Proof of the Lemma 2.2 (continued):

Let p(n) denote the number of pairs of the possible internal configurations of O which has the input head placed just behind the symbol 2.

$p(n) = \binom{K}{2} + K$, where $K = r.s^{S(2n+1)} . S(2n+1)$

$p(n) \leq c^{S(2n+1)}$, where c is some appropriate constant.

On the other hand, the number of all elements of V(n) is

$|V(n)| = 2^n$

The fact that $\lim_{n \to \infty}(S(n)/n) = 0$ implies that there is a positive integer n_0 such that $\forall_{n \geq n_0} : p(n) < |V(n)|$

Therefore for any large n there must be two different strings x and y in V(n) such that $C(x) \cap C(y) \neq \emptyset$. This contradicts the Proposition 2.3. Consequently there is no 1 sutm (S(n))

which recognizes L_1 for any S(n) such that $\lim_{n\to\infty} (S(n)/n) = 0$. □

Theorem 2.4: For any function S(n) such that $\lim_{n\to\infty} (S(n)/n)=0$ the next assertions are true:

1. 1 SUTM (S(n)) $\subsetneq$ 1 SATM (S(n))
2. if also S(n) $\geq$ log n then 1 SUTM (S(n)) $\subsetneq$ SUTM (S(n))

Lemma 2.5: Let $L_2 = \{w2w \;/\; w \in \{0,1\}^+\}$. Then

1. $L_2 \in$ 1 SUTM (1)
2. $\overline{L_2} \notin$ 1 SUTM (S(n)) for any S(n) such that $\lim_{n\to\infty} (S(n)/n)=0$.

Theorem 2.6: For any function S(n) such that $\lim_{n\to\infty}(S(n)/n)=0$, the class of languages recognized by S(n)-space bounded one--way synchronized alternating Turing machines with only universal states is not closed under complementation.

The next table summarizes some preceding results.

S(n) \ classes	1 SUTM(S(n)) -1SATM(S(n))	1 SUTM(S(n)) -SUTM(S(n))	1 ATM(S(n)) -1SATM(S(n))	1 UTM(S(n)) -1SUTM(S(n))
log log n	$\subsetneq$	?	$\subsetneq$	$\subsetneq$
log n	$\subsetneq$	?	?	?
n	$\subsetneq$	$\subsetneq$	?	?

Notation: ntm (S(n)) denote the S(n)-space bounded nondeterministic Turing machine.

Lemma 2.7: Let S(n) be the space-constructible function

1. For any utm (S(n)) M there is an ntm(S(n)) N such that $L(N) = \overline{L(M)}$.
2. For any ntm (S(n)) O there is an utm (S(n)) P such that $L(P) = \overline{L(O)}$.

Lemma 2.8 |17|: For any space-constructible function $S(n):N\to R$, such that $S(n)\geq \log n$, the class NTM(S(n)) is closed under complementation.

Theorem 2.9: Let S(n) be a space constructible function such that S(n) $\geq$ log n. Then

UTM (S(n)) = NTM (S(n)).

References

|1| A.K.Chandra, D.C.Kozen and L.J.Stockmeyer, Alternation J.of ACM 28 (1981) 114-133.

|2| E.M.Gurari and O.H. Ibarra, (Semi-)alternating stack automata, Math. System Theory 15 (1982) 211-224.

|3| J.E.Hopcroft and J.D.Ullman, Formal languages and their relation to automata (Addison-Wesley, Reading, MA, 1969).

|4| J.Hromkovič, Alternating multicounter machines with constant number of reversals, Information Processing Letters 21 (1985) 7-9.

|5| J.Hromkovič, On the power of alternation in automata theory, J.of Comp. and Sys. Sci. 31 (1985) 28-39.

|6| J.Hromkovič, Tradeoffs for language recognition on parallel computing models, Proc. 13th ICALP '86, Lecture Notes in Computer Science 226 (1986) 157-166.

|7| K.Inoue, A.Ito, I.Takanami and H.Taniguchi, A space-hierarchy result on two-dimensional alternating Turing machines with only universal states, Inform. Sciences 35 (1985) 79-90.

|8| K.Inoue, A.Ito, I.Takanami and H.Taniguchi, Two-dimensional alternating Turing machines with only universal states, Inform. and Control 55 (1982) 193-221.

|9| K.Inoue, H.Matsuno, I.Takanami and H.Taniguchi, Alternating simple multihead finite automata, Theoret. Comp. Sci. 36 (1985) 291-308.

|10| K.Inoue, I.Takanami and R.Vollmar, Alternating on-line Turing machines with only universal states and small space bounds, Theoret. Comp. Sci. 41 (1985) 331-339.

|11| K.N.King, Alternating finite Automata, Doctoral Dissertation, University of California, Berkeley.

|12| K.N.King, Alternating multihead finite automata, Proc. 8th ICALP '81, Lecture Notes in Computer Science 115 (1981) 506-520.

|13| R.J.Ladner, R.J.Lipton and L.J.Stockmeyer, Alternating pushdown and stack automata, SIAM J. Comput. 13 (1984) 135-155.

|14| W.J.Paul, E.J.Prauss and R.Reischuk, On alternation, Acta Informatica 14 (1980) 243-255.

|15| W.J.Paul and R.Reischuk, On alternation II., Acta Informatica 14 (1980) 391-403.

|16| A.Slobodová, On the power of communication in alternating machines, submitted to MFCS '88.

|17| R.Szelepcsenyi, The method for forcing for nondeterministic automata, EATCS Bulletin (1987).

IMYCS'88, Smolenice Castle, November 14-18, 1988.

ALGEBRAIC DISCRETE FOURIER TRANSFORMS AND FAST CONVOLUTION ALGORITHMS

GABRIELE STEIDL
Wilhelm-Pieck-Universität Rostock
Sektion Mathematik
Universitätsplatz 1
Rostock, 2500
DDR

1. INTRODUCTION. The reduced number of multiplications and the fact that no round-off errors occur in calculating cyclic convolutions via generalized discrete Fourier transforms (GFT's) over finite commutative rings R with identity are the main reasons for the great attention that these techniques have received over the last few years. To overcome one drawback of the GFT, the rigid relation between obtainable transform length and the ring in which the transform is defined, several authors [2]-[4] consider extension rings of R.

We introduce extension rings S of R supporting a GFT of given length N. Based on a normal basis concept in S, we generalize the algebraic discrete Fourier transform (ADFT) [1] for rings R and present a convolution algorithm via ADFT. We show the connection between this algorithm and other convolu-

tion techniques via Chinese remainder theorem (CRT) or via reduced GFT's. The ADFT has the advantage over these transforms that its inverse can be calculated by a similar algorithm. This property of the ADFT facilitates the soft- and hardware realization of cyclic convolutions of R-sequences.

2. GENERALIZED DISCRETE FOURIER TRANSFORMS. Throughout this paper, let Z be the ring of all integers, and let R be a finite commutative ring with identity. By R^*, we denote the group of units in R. For $N \in Z$ ($N \geq 2$), an element $e \in R$ is called a primitive N-th root of unity in R if

$$e^N = 1, \quad e^k - 1 \in R^* \quad (k = 1,\ldots,N-1).$$

If for given $N \in Z$ ($N \geq 2$), there exists a primitive N-th root of unity in R, then R supports a GFT of length N. In this case, the GFT of length N over R and its inverse are defined to be the following mappings between $\underline{y} = [y_i]_{i=0}^{N-1} \in R^N$ and $\hat{\underline{y}} = [\hat{y}_j]_{j=0}^{N-1} \in R^N$ [2]:

$$\hat{y}_j := \sum_{i=0}^{N-1} y_i \, e^{ij} \quad (j = 0,\ldots,N-1),$$

$$y_i := N^{-1} \sum_{j=0}^{N-1} \hat{y}_j \, e^{-ij} \quad (i = 0,\ldots,N-1).$$

The GFT possesses properties resembling those of the classical discrete Fourier transform (DFT), particularly the cyclic convolution property [3]. The advantage of the GFT over the DFT is that one can replace complex arithmetic by operations in R, for example by the residue arithmetic modulo $M \in Z$ ($M \geq 2$) if $R := Z/MZ$. The main drawback of the GFT results from the following relation between obtainable transform

length and the ring, in which the transform is defined:

Assume that R can be decomposed into the direct sum of r local rings with the corresponding residue fields $GF(q_k)$

(1) $R \cong \bigoplus_{k=1}^{r} R_k$.

Then R supports a GFT of length N if and only if $N \mid q_k - 1$ for all $k = 1,\ldots,r$.

Recently [5], we have showed how to construct extension rings S of a given ring R supporting a GFT of length N:

Let R be given by (1), and let $N \in Z$ ($N \geq 2$) with

(2) $\gcd(q_k,N) = 1$ $(k = 1,\ldots,r)$.

We consider the elements of the ring Z/NZ as the integers $0,\ldots,N-1$, where addition and multiplication is performed modulo N. Let

(3) $U = \langle q_1,\ldots,q_r \rangle \leq (Z/NZ)^*$

be the subgroup of $(Z/NZ)^*$ generated by all q_k, where $n := |U|$. There exists a primitive N-th root of unity e_k in a Galois extension of R_k of degree $d_k := \mathrm{ord}_N(q_k)$. Further,

$$f^{(k)}(x) := \prod_{u \in U} (x - e_k^u) \quad (k = 1,\ldots,r)$$

are monic polynomials in $R_k[x]$. With respect to

$$R[x] \cong \bigoplus_{k=1}^{r} R_k[x]$$

define the monic polynomial $f \in R[x]$ of degree n by

(4) $f \longleftrightarrow (f^{(1)},\ldots,f^{(r)})$.

In this way, one can obtain $(\varphi(N)/n)^r$ pairwise different polynomials $f \in R[x]$, where φ denotes the Euler's totient function. With $(f) = f \cdot R[x]$, set

(5) $S := R[x]/(f)$.

The elements of S can be represented as polynomials in R[x]

of degree $< n$, where all calculations are performed modulo f.

Now $x \in S$ is a primitive N-th root of unity in S, and the GFT of length N over S and its inverse are defined by

(6) $\quad \hat{\underline{y}} = A_N \underline{y}, \ \underline{y} = A_N^{-1} \hat{\underline{y}},$

where $\underline{y}, \hat{\underline{y}} \in S$ and

$$A_N := [x^{ij}]_{i,j=0}^{N-1}, \ A_N^{-1} := N^{-1} [x^{-ij}]_{i,j=0}^{N-1}.$$

3. CRT AND REDUCED GFT. We use the notations (1)-(5). For $i,j \in Z/NZ$, we introduce an equivalence relation $i \sim j$, if there exists $u \in U$ such that $i = uj$. Let

(7) $\quad T = \{t_k : k = 1,\ldots,m\}$

be a set of representatives of the equivalence classes of Z/NZ induced by U. Assume that $e \in S$ with $f(e) = 0$. Then $x^N - 1 \in R[x]$ factors over S

$$x^N - 1 = \prod_{k=1}^{m} f_k(x), \ f_k(x) = \prod_{j \sim t_k} (x - e^j) \ (k = 1,\ldots,m),$$

and it holds by the CRT that

$$R[x]/(x^N - 1) \cong \bigoplus_{k=1}^{m} R[x]/(f_k).$$

One can generalize the algorithm [4, pp.35-37] for the convolution of N-point sequences in R in the obvious manner [6].

Another method to calculate such convolutions is based on the GFT (6) over S. Unfortunately, this approach extends the R-arithmetic to the more expensive arithmetic in S. One compromise makes use of the so-called <u>conjugate symmetry property</u> [2]: There exists an automorphism group of S over R given by $\mathrm{Aut}_R S := \{\sigma_u : u \in U\}$, $\sigma_u(x) := x^u \in S$. Now it holds for all $u \in U$ and for $\underline{y} \in R^N$ by (6) that

$$\sigma_u(\hat{y}_j) = \sum_{i=0}^{N-1} y_i \, x^{iju} = \hat{y}_{ju}.$$

Hence, the GFT of $\underline{y} \in R^N$ is determined by $[\hat{y}_{t_k}]_{k=1}^m$. The GFT, which calculates exactly the representatives $\hat{y}_{t_k}$ ($k = 1,\ldots, m$) of the components of $\hat{\underline{y}}$ is called the reduced GFT. A convolution algorithm via reduced GFT was developed in [3]. In [6], we explain the connection between this algorithm and those based on the CRT.

However, with respect to soft- and hardware realizations of convolutions, both algorithms have the disadvantage that the applied transform and its inverse must be calculated in a different manner.

4. THE ALGEBRAIC DISCRETE FOURIER TRANSFORM. We use the notations (1)-(7). If $B := \{\sigma_u(b): u \in U\}$ ($b \in S$) contains n linearly independent elements over R, then B is a normal basis of S over R. By [5], there exists a normal basis of S over R.

The linear map of S onto R defined by

$$\mathrm{tr}(a) := \sum_{u \in U} \sigma_u(a) \quad (a \in S)$$

is called the trace of $a \in S$ over R. Two bases $B = \{b_1,\ldots,b_n\}$ and $C = \{c_1,\ldots,c_n\}$ of S over R are dual bases if $\mathrm{tr}(b_i c_j) = \delta_{ij}$, where δ_{ij} signifies the Kronecker symbol. If B = C, then B is a self-dual basis of S over R. By [5], for any basis $B = \{b_1,\ldots,b_n\}$ of S over R, there exists a unique dual basis $C = \{c_1,\ldots,c_n\}$ determined by

$$[c_i]_{i=1}^n = D_B^{-1} [b_i]_{i=1}^n, \quad D_B := [\mathrm{tr}(b_i b_j)]_{i,j=1}^n,$$

which is a normal basis of S over R if and only if B is a normal basis.

Let B be a normal basis of S over R with the dual basis C.

Now we generalize the ADFT [1, p.252] for finite commutative rings with identity. The ADFT of length N over R and its inverse are defined to be the following mappings between N-point vectors $\underline{y} = [y_i]_{i=0}^{N-1} \in R^N$ and $\underline{Y} = [Y_j]_{j=0}^{N-1} \in R^N$;

$$(8) \quad \underline{Y} := B_{A_N}\, \underline{y}, \quad \underline{y} := C_{A_N^{-1}}\, \underline{Y},$$

with

$$B_{A_N} := [(x^{ij})_1]_{i,j=0}^{N-1}, \quad C_{A_N^{-1}} := N^{-1}\, [[x^{-ij}]_1]_{i,j=0}^{N-1},$$

where $(x^{ij})_1$ denotes the coefficient of $\sigma_1(b) = b$ in the representation of $x^{ij} \in S$ with respect to B, and where $[x^{-ij}]_1$ is the coefficient of $\sigma_1(c)$ in the representation of $x^{-ij} \in S$ with respect to C.

Theorem [6]. Under above assumptions, the ADFT of $\underline{y} \in R^N$ calculates $[\hat{y}_{t_k}]_{k=1}^m$ with respect to the normal basis B.

With other words, $[\hat{y}_{t_k}]_{k=1}^m$ can be arranged from $\underline{Y}$, and conversely.

For given $\underline{y}, \underline{z} \in R^N$, the cyclic convolution $\underline{h} = \underline{y} * \underline{z}$ can be obtained via ADFT as follows:

1) Calculate $\underline{Y}, \underline{Z} \in R^N$ via ADFT and arrange $[\hat{y}_{t_k}]_{k=1}^m$, $[\hat{z}_{t_k}]_{k=1}^m$.
2) Evaluate $\hat{h}_{t_k} = \hat{y}_{t_k} \cdot \hat{z}_{t_k}$ for all $k = 1, \ldots, m$.
3) Arrange $\underline{H} \in R^N$ and calculate $\underline{h} \in R^N$ via inverse ADFT.

5. CONCLUSIONS. The ADFT over R is defined for any transform length N satisfying (2). For example, one can perform ADFT's of large length over residue class rings of integers modulo Fermat- or Mersenne numbers. The ADFT uses only R-arithmetic and can be computed by fast algorithms for some R and N [1, 6]

Its main advantage over the transforms considered in Sec-

tion 3 is that its inverse can be performed by a similar algorithm. By (8), it is desirable to choose B as a self-dual normal basis of S over R (cf. [1, 6]).

The ADFT (inverse ADFT) can be calculated by the reduced GFT or by the CRT and by postconverting the outputs according to the normal basis B (C). On the other hand, the normal basis concept yields a possibility to perform reduced GFT's or transforms based on the CRT and their inverses via same algorithms at the cost of precomputing the input data for the inverse transforms with respect to B and postcomputing the output values with respect to C.

REFERENCES

[1] T. Beth, Verfahren zur schnellen Fourier-Transformation (Stuttgart, 1984).

[2] E. Dubois and A.N. Venetsanopoulos, Convolutions using a conjugate symmetry property for the generalized discrete Fourier transform, IEEE ASSP 26/2(1978) 165-170.

[3] J.B. Martens and M.C. Vanwormhoudt, Convolutions of long integer sequences by means of number theoretic transforms over residue class polynomial rings, IEEE ASSP 31/5(1983) 1125-1134.

[4] H.J. Nussbaumer, Fast Fourier Transforms and Convolution Algorithms (Berlin, 1980).

[5] G. Steidl, On normal bases for finite commutative rings, submitted for publication.

[6] G. Steidl, Generalized algebraic Fourier transforms and convolution algorithms, submitted for publication.

IMYCS'88, Smolenice Castle, November 14-18, 1988.

A New Approach to Verification of Programs with Higher-Order Arrays
EXTENDED ABSTRACT

Research supported by a grant no RP.I.09 of Polish Ministry of National Education.

JERZY TYSZKIEWICZ

Institute of Mathematics,
Zakład Logiki Matematycznej,
University of Warsaw,
PKiN, 00-901 Warszawa,
Poland.

§0 INTRODUCTION

The higher-order arrays, a model of powerful data structures were first defined by J. Tiuryn in the paper [4]. They generalizes the notion of usual "level 1" arrays, considered e.g. in [6]. A higher- order array is a partial function which arguments and values are rather arrays of lower levels than the ground domain objects. In the papers [4], [6], [2], [1] and others the problems of decidability of halting problem over finite structures, verification and expressive power of programs with higher-order arrays were considered. In our paper we assume a new notion of semantics for programs equipped with higher-order arrays. Our main aim is to obtain a possibility to verify partial correctness of these programs with only first-order ground domain theory oracles. The general result is negative, although we obtain positive partial results. The main tool in our considerations is the complexity theory.

§1 SYNTAX

Let L be an arbitrary, finite, first-order language with equality. The language L^1 of *higher-order arrays* is a disjoint sum of L and the set $\{\perp_\sigma, \circ_{(\sigma\to\tau)} | \sigma, \tau \in \mathbf{TYPE}\}$, where **TYPE** is the least set containing the *ground type* 0 and closed under the followig rule: if $\sigma, \tau \in \mathbf{TYPE}$ then $(\sigma \to \tau) \in \mathbf{TYPE}$. For $\sigma \in \mathbf{TYPE}$ we have sets of *variables* (typically X) and *array identifiers* (typically A) of type σ. For each $\sigma \in \mathbf{TYPE}$ we define the set $\mathbf{T}_\sigma$ of *terms* of type σ as the least set satisfying:

* $\perp_\sigma$ is a term of type σ;

* a variable, an array identifier and a function symbol from L of type σ are terms of type σ;
* if $T \in \mathbf{T}_{(\sigma\to\tau)}$ and $S \in \mathbf{T}_\sigma$ then $T \circ_{(\sigma\to\tau)} S$ is a term of type σ.
Informal meaning of $T \circ_{(\sigma\to\tau)} S$ is an application of a function T to an argument S. Informal meaning of a symbol $\bot_0$ is an undefined value. From now on we will omit types and application symbols $\circ_{(\sigma\to\tau)}$ if it makes no confusions. A term T of type σ is *closed* if it contains no variables (notation $T \in \mathbf{CT}_\sigma$). It is *assignable* if closed, of type 0 and of the form $AT_1 \dots T_n$, where A is an array identifier and $T_1 \dots T_n$ are of appropriate types. We also allow the case of $n = 0$. The set **ASSN** of *assertions* we define as the least set satisfying:
* if $T_1, T_2 \in \mathbf{T}_\sigma$ then $T_1 = T_2 \in$ **ASSN**;
* if r is a relation symbol of arity k from L and $T_1 \dots T_n \in \mathbf{T}_0$ then $r(T_1 \dots T_n) \in$ **ASSN**;
* if $\alpha, \beta \in$ **ASSN**and X is a variable then $\neg\alpha, \alpha \wedge \beta$ and $(\exists X)\alpha$ are also in **ASSN**.
Abbreviations $\Rightarrow, \vee, \forall, \Leftrightarrow$ are defined as usually. Let $\mathbf{ASSN}_0$ be the set of all $\alpha \in$ **ASSN** such that α contains only variables and array identifiers of type 0. We identify formulae of $\mathbf{ASSN}_0$ with first-order formulae over $L^+ = L \cup \{\bot_0\}$. The set **WPA** of *while-programs with higher-order arrays* is the least set such that:
* if T is an assignable term and $S \in \mathbf{CT}_0$ then an *assignment* $T := S \in$ **WPA**;
* If $P, Q \in$ **WPA** and $e \in$ **ASSN** is closed, without quantifiers and equalities between terms of type different than 0 then expressions **if** e **then** P **else** Q **fi**, $(P;Q)$, **while** e **do** P **od** are in **WPA**.
We assume that expressions of the form $[P]\alpha$ for $P \in$ **WPA** and $\alpha \in$ **ASSN** are formulae (but not assertions). We extend this convention for all elements of the closure of assertions and formulae of the form $[P]\alpha$ under $\wedge$ and $\neg$. Particulary $\alpha \Rightarrow [P]\beta$ is a formula. The second, well-known equivalent notation for $\alpha \Rightarrow [P]\beta$ is $\{\alpha\}P\{\beta\}$.

§2 SEMANTICS

An *$L^\sharp$-structure* $\mathcal{B}$ is a family of disjoint sets $\{\mathcal{B}_\sigma | \sigma \in \mathbf{TYPE}\}$, satisfying the following conditions:
* $\mathcal{B}_0$ is an L^+-structure in the standard first-order sense;
* $\mathcal{B}_{(\sigma\to\tau)}$ is a subset of the space $\mathbf{FUN}(\mathcal{B}_\sigma, \mathcal{B}_\tau)$ of all total, set-theoretical functions from $\mathcal{B}_\sigma$ into $\mathcal{B}_\tau$, containing a constant function equal $\bot_\tau$ for all arguments, which is an interpretation of the symbol $\bot_{(\sigma\to\tau)} \in L$.
* $\mathcal{B}_{(\sigma\to\tau)}$ is closed under finite modifications of its elements, i.e. if $card(\{x \in \mathcal{B}_\sigma : f(x) \neq g(x)\}) < \aleph_0$ and $f \in \mathcal{B}_{(\sigma\to\tau)}$ then $g \in \mathcal{B}_{(\sigma\to\tau)}$;
* an interpretation of $\circ_{(\sigma\to\tau)}$ is a function from the Cartesian product $\mathcal{B}_{(\sigma\to\tau)} \times \mathcal{B}_\sigma$ into $\mathcal{B}_\tau$ such that for $f \in \mathcal{B}_{(\sigma\to\tau)}$ and $x \in \mathcal{B}_\sigma$, $f \circ x = f(x) \in \mathcal{B}_\tau$.
The definition of satisfiability for all formulae in such a structure is standard. This relation is denoted $\mathcal{B} \models \alpha$. An *$L^\sharp$-polystructure* is an arbitrary, nonempty class $\mathcal{M}$ of $L^\sharp$-structures, equipped with relations $\models$. We say that a formula α (not necessary an assertion) is true in $\mathcal{M}$ and write $\mathcal{M} \models \alpha$ iff $\mathcal{B} \models \alpha$ for all $\mathcal{B} \in \mathcal{M}$. A *theory* of $\mathcal{M}$, $Th(\mathcal{M})$, is the set $\{\alpha | \alpha \in$ **ASSN**$, \mathcal{M} \models \alpha\}$ and a *partial correctness theory* of $\mathcal{M}$ is the set $PC(\mathcal{M}) = \{\alpha \Rightarrow [P]\beta | \alpha, \beta \in$ **ASSN**$, P \in$ **WPA**$, \mathcal{M} \models \alpha \Rightarrow [P]\beta\}$. For given L^+-structure $\mathcal{A}$ we define the *first-order theory* of $\mathcal{A}$ as the set $Th(\mathcal{A}) = \{\alpha | \alpha \in$ **ASSN**$, \mathcal{B} \models \alpha\}$ for any $L^\sharp$-polystructure $\mathcal{B}$ such that $\mathcal{B} = \mathcal{A}$. It is easy to see that

$Th(\mathcal{A})$ is well-defined and coincides with the standard notion of theory for first-order structures.

§3 EXPRESSIVENESS AND WEAK EXPRESSIVENESS

We say that an $L^{\natural}$-polystructure $\mathcal{M}$ is *weakly expressive for* $L^{\natural}$ *and* **WPA** iff for each $P \in$ **WPA** and each $\beta \in$ **ASSN** there exists an $\alpha \in$ **ASSN** satisfying:

* $\mathcal{M} \models \alpha \Rightarrow [P]\beta$;

* for an arbitrary $Q \in$ **WPA** and an arbitrary $\eta \in$ **ASSN** if $\mathcal{M} \models \eta \Rightarrow [Q;P]\beta$ then $\mathcal{M} \models \eta \Rightarrow [Q]\alpha$.

An $L^{\natural}$-polystructure $\mathcal{M}$ is said to be *expressive for* $L^{\natural}$ *and* **WPA** iff for each $P \in$ **WPA** and each $\beta \in$ **ASSN** there exists an $\alpha \in$ **ASSN** satisfying $\mathcal{M} \models \alpha \Leftrightarrow [P]\beta$.

The notion of weak expressiveness is due to Jerzy Tiuryn [4]. It is easy to see that expressiveness implies weak expressiveness for an arbitrary $L^{\natural}$-polystructure $\mathcal{M}$.

§4 VERIFICATION OF PROGRAMS WITH HIGHER-ORDER ARRAYS

Now we describe a Hoare-like system **H** for programs with higher-order arrays and assertions being first-order formulae over the language $L^{\natural}$.

THE SYSTEM H(W)

ORACLE

$$W \subseteq \textbf{ASSN}$$

ASSIGNMENT AXIOM

$$\alpha\{T/S\} \Rightarrow [S := T]\alpha$$

R1 COMPOSITION RULE

$$\frac{\alpha \Rightarrow [P]\beta \quad \beta \Rightarrow [Q]\gamma}{\alpha \Rightarrow [P;Q]\gamma}$$

R2 IF-THEN-ELSE RULE

$$\frac{\alpha \wedge e \Rightarrow [P]\beta \quad \alpha \wedge \neg e \Rightarrow [Q]\beta}{\alpha \Rightarrow [\textbf{if } e \textbf{ then } P \textbf{ else } Q \textbf{ fi}]\beta}$$

R3 WHILE RULE

$$\frac{\alpha \wedge e \Rightarrow [P]\alpha}{\alpha \Rightarrow [\textbf{while } e \textbf{ do } P \textbf{ od}]\alpha \wedge \neg e}$$

R4 CONSEQUENCE RULE

$$\frac{\alpha \Rightarrow \beta \quad \beta \Rightarrow [P]\gamma \quad \gamma \Rightarrow \eta}{\alpha \Rightarrow [P]\eta}$$

4.1 Theorem

For an arbitrary formula $\alpha \Rightarrow [P]\beta$ and an arbitrary $L^{\natural}$-polystructure $\mathcal{M}$ the following are equivalent:

a) $\mathcal{M} \models \alpha \Rightarrow [P]\beta$

b) $\mathbf{H}(Th(\mathcal{M})) \vdash \alpha \Rightarrow [P]\beta$.

The proof of this is very standard and can be found e.g. in [4].

§5 PROPERTIES OF CHOSEN FAMILIES OF $L^{\natural}$-POLYSTRUCTURES

* $\mathcal{B}_{(\sigma\to\tau)} = \{f \in \mathbf{FUN}(\mathcal{B}_\sigma, \mathcal{B}_\tau) | f(x) = \perp_\tau$ for almost all elements $x \in \mathcal{B}_\sigma\}$.

5.3.1 Theorem (Kowalczyk, Urzyczyn, [3])
a) For an arbitrary $\mathcal{A}$, $\mathcal{M}_3(\mathcal{A})$ is expressive.
b) If $\mathcal{A}$ is an infinite L^+-structure then $Th(\mathcal{M}_3(\mathcal{A}))$ is not an arithmetic set.

We say that an L^+-structure $\mathcal{A}$ is *strongly arithmetical* iff there exists a bijection *code* : $\mathcal{A} \to \omega$ identifying elements of $\mathcal{A}$ with natural numbers such that arithmetic on codes may be defined by formulae of L^+ : $Zero(x)$, $Succ(x,y)$, $Add(x,y,z)$, $Mult(x,y,z)$, which may depend on $\mathcal{A}$. By a result of Urzyczyn [7] it is equivalent to the followig condition: $\mathcal{A}$ is an infinite, Herbrand structure, expressive for class of while-programs with recursion.

5.3.2 Theorem (Kowalczyk, Urzyczyn, [3])
a) There exists a finitary, sound and relatively complete system **G** for $PC(\mathcal{M}_3(\mathcal{A}))$ using $Th(\mathcal{A})$ as an oracle, for all strongly arithmetical or finite structures $\mathcal{A}$.
b) There is no Turing machine **T** with oracle which for an arbitrary weakly expressive $\mathcal{M}_3(\mathcal{A})$ accepts exactly $PC(\mathcal{M}_3(\mathcal{A}))$ using oracle $Th(\mathcal{A})$.

5.4
Let, for a given L^+-structure $\mathcal{A}$, $\mathcal{M}_4(\mathcal{A})$ be a family of all $L^\sharp$-structures $\mathcal{B}$ satisfying condition that $\mathcal{B}_0$ is elementary equivalent to $\mathcal{A}$.

5.4.1 Theorem
a) For an arbitrary $\mathcal{A}$, $Th(\mathcal{M}_4(\mathcal{A}))$ is Σ^0_1 over $Th(\mathcal{A})$.
b) There exists a finitary system **G** (which is **H** with added new axioms and rules) such that **G** is sound and relatively complete for each weakly expressive $\mathcal{M}_4(\mathcal{A})$.
c) If $\mathcal{A}$ is a finite structure then $\mathcal{M}_4(\mathcal{A})$ is expressive.
d) $\mathcal{M}_4(\mathcal{N})$ is not weakly expressive.
Sketch of proof:
a) We can axiomatize $Th(\mathcal{M}_4(\mathcal{A}))$ taking as axioms whole $Th(\mathcal{A})$ and adding new axiom schemes which force a good "type inference", functional "behaviour" of sorts $\mathcal{B}_\sigma$ for $\sigma \neq 0$, existence of constants $\perp_{(\sigma\to\tau)}$ and possibility of finite modification of functions. We must also add Modus Ponens and Generalization Rules.
b) Follows from a) and 2.1.
c) It is standard.
d)Let

$$P \equiv y := 0;\ \textbf{while}\ y \neq x\ \textbf{do}\ y := y + 1\ \textbf{od}.$$

Assume that $\mathcal{M}_4(\mathcal{N})$ is weakly expressive. Then there exists an $\alpha(x) \in$ **ASSN** such that $\mathcal{M}_4(\mathcal{N}) \models \alpha \Rightarrow [P]false$ and for each $\beta(x) \in$ **ASSN** if $\mathcal{M}_4(\mathcal{N}) \models \beta \Rightarrow [P]false$ then $\mathcal{M}_4(\mathcal{N}) \models \alpha \Rightarrow \beta$. Take

$$\beta(x) \equiv \alpha(x) \vee (\exists Y : (0 \to \tau))(\forall z)(Y0 = \perp \wedge (Yz = \perp \Rightarrow Y(z+1) = \perp) \wedge Yx \neq \perp),$$

where τ is a type longer than all types of variables and array identifiers which occur in α. It is clear that $\mathcal{M}_4(\mathcal{N}) \models \beta \Rightarrow [P]false$. It is also a simple observation that $\mathcal{M}_4(\mathcal{N}) \not\models \alpha \Rightarrow \beta$, which yields a contradiction.

In this paragraph we consider four functions of the form (L^+-structure $\mathcal{A}$)$\mapsto$($L^\sharp$-polystructure $\mathcal{M}(\mathcal{A})$). Intended meaning of such a function is the following: given $\mathcal{A}$;then $\mathcal{M}(\mathcal{A})$ is a family of $L^\sharp$-structures $\mathcal{B}$ such that $\mathcal{B}_0$ is "similar" to $\mathcal{A}$ in some sense, and $\mathcal{B}_\sigma$, for $\sigma \neq 0$, satisfy some additional conditions. This scheme reflects the fact that in ordinary life arrays are not used for their own, but are treated as tools in construction of programs operating on the ground type objects. In particulary we want to find "good" such a functions. By a good function $\mathcal{A} \mapsto \mathcal{M}(\mathcal{A})$ we mean such a function that $\mathcal{M}(\mathcal{A})$ is at least weakly expressive for $\mathcal{A}$ being finite or a standard model of arithmetics. Secondly, we want to have, for weakly expressive $\mathcal{M}(\mathcal{A})$, a finitary, sound and relatively complete system G for $PC(\mathcal{M}(\mathcal{A}))$ with only $Th(\mathcal{A})$ as an oracle. Remember that if we allow $Th(\mathcal{M}(\mathcal{A}))$ as an oracle then the wanted system always exists by 4.1. We assume that a good function should be also natural in some sense.
Convention: $\mathcal{N} = (\omega, +, \cdot, 0, 1)$ denotes the standard model of arithmetics.

5.1
Let $\mathcal{M}_1(\mathcal{A})$ be a singleton $\{\mathcal{B}\}$, where $\mathcal{B}$ is the following $L^\sharp$-structure:
* $\mathcal{B}_0 = \mathcal{A}$;
* $\mathcal{B}_{(\sigma\to\tau)} = \mathbf{FUN}(\mathcal{B}_\sigma, \mathcal{B}_\tau)$.

5.1.1 Theorem
a) If $\mathcal{A}$ is finite then $\mathcal{M}_1(\mathcal{A})$ is expressive.
b) $\mathcal{M}_1(\mathcal{N})$ is expressive.
c) Complexity of $Th(\mathcal{M}_1(\mathcal{N}))$ is the theory of all finite types of the standard model of arithmetics.
d) There is no Turing machine T with oracle which for an arbitrary weakly expressive $\mathcal{M}_1(\mathcal{A})$ accepts exactly $PC(\mathcal{M}_1(\mathcal{A}))$ using oracle $Th(\mathcal{A})$.
The proof of it is very standard.

5.2
Let $\mathcal{M}_2(\mathcal{A})$ be, for a given $\mathcal{A}$, a family of all $L^\sharp$-structures $\mathcal{B}$ such that $\mathcal{B}_0 = \mathcal{A}$.

5.2.1 Theorem
a) If $\mathcal{A}$ is finite then $\mathcal{M}_2(\mathcal{A})$ is expressive.
b) $\mathcal{M}_2(\mathcal{N})$ is expressive.
c) $Th(\mathcal{M}_2(\mathcal{N}))$ is Π^1_1-hard.
d) There is no Turing machine T with oracle which for an arbitrary weakly expressive $\mathcal{M}_2(\mathcal{A})$ accepts exactly $PC(\mathcal{M}_2(\mathcal{A}))$ using oracle $Th(\mathcal{A})$.
Sketch of proof:
a) and b) are again standard and d) follows from c). To prove c) observe that that $\mathcal{M}_1(\mathcal{N}) \subseteq \mathcal{M}_2(\mathcal{N})$ and all elements $\mathcal{B} \in \mathcal{M}_2(\mathcal{N})$ are substructures of $\mathcal{B}^0$-the only element of $\mathcal{M}_1(\mathcal{N})$. It follows that theories of $\mathcal{M}_1(\mathcal{N})$ and $\mathcal{M}_2(\mathcal{N})$ are identical w.r.t. Π^1_1-sentences.

5.3
Let, for a given $\mathcal{A}$, $\mathcal{M}_3(\mathcal{A})$ be a singleton $\{\mathcal{B}\}$, where $\mathcal{B}$ is the following $L^\sharp$-structure:
* $\mathcal{B}_0 = \mathcal{A}$;

§6 CONCLUSIONS AND REMARKS

6.1 Theorem

If $1 \leq i, j \leq 4$ and $i \neq j$ then there exists a language L and an L^+-structure $\mathcal{A}$ such that $PC(\mathcal{M}_i(\mathcal{A})) \neq PC(\mathcal{M}_j(\mathcal{A}))$.

Sketch of proof:

Complexity bounds for $\mathcal{A} = \mathcal{N}$ give us all distinctions except that $PC(\mathcal{M}_1(\mathcal{N})) \neq PC(\mathcal{M}_2(\mathcal{N}))$. For to prove this observe that

$$Inf \equiv (\exists X : (0 \to 0))(\forall z) X z \neq \bot$$

is true in $\mathcal{M}_1(\mathcal{N})$ while not in $\mathcal{M}_2(\mathcal{N})$. It follows that the partial correctness theories of these polystructures are distinct, too.

As it may be seen from the above theorems, situation is not satisfactory. It is the result of our assumption allowing as assertions arbitrary first-order formulae over $L^\natural$, i.e., in fact, higher-order formulae over L. This forces the partial correctness theory to be very complicated (to be of very high complexity). Therefore I suppose that we may not expect a success (a sound and relatively complete system for PC with only first-order ground domain oracles) allowing such a strong set of assertions.

The reader is referred to the paper [5] for discussion of our problem from a different point of view. The author uses only first-order Σ_1 assertions, but works not in the ground domain, but in the weak version of Set Theory (so called KPU) built on it. He obtains a sound and relatively complete system and nontrivial class of expressive structures.

Acknowledgment

I wish to thank Jerzy Tiuryn for his inspiring suggestions and comments.

§7 REFERENCES

[1] Kfoury,A.J.,Urzyczyn,P.
Finitely typed functional programs, Part II *to appear.*

[2] Kfoury,A.J.,Tiuryn,J.,Urzyczyn,P.
The hierarchy of simply typed functional programs *Proc. LICS'87.*

[3] Kowalczyk,W.,Urzyczyn,P.
Verification of programs with higher-order arrays *Proc. FCT'87.*

[4] Tiuryn,J.
Higher-order arrays and stacks in programming. An application of complexity theory to logics of programs *Proc. MFCS'86.*

[5] Tiuryn,J.
A simple programming language with data types: semantics and verification *Proc. Logic of programs'85, LNCS, vol. 193, 1985, 387-405.*

[6] Tiuryn,J.,Urzyczyn,P.
Some relationships between logics of programs and complexity theory *Proc. FOCS'83, to appear in Theoretical Computer Science.*

[7] Urzyczyn, P.
A necessary and sufficient condition in order that a Herbrand interpretation be expressive relative to recursive programs *Inf. Control, vol.56, 1983, 212-219.*

IMYCS'88, Smolenice Castle, November 14-18, 1988.

The Convex Hull Problem on Grids - Computational and Combinatorial Aspects (Extended Abstract)

Kristel Unger
Karl-Weierstraß-Institut für Mathematik
Mohrenstr. 39
Berlin, DDR-1086

1. Introduction

For some years research in the field of computational geometry has became more intensive. First of all, this is caused by increased applications. We refer to computer graphics, CAD/CAM at all, digital image processing and layout verification as fields of application. A great manifold of articles, but also of monographs, e.g. [1] , are proofs of this intensive interest. In [1] one finds an extensive list of references.

One of the most investigated problems is the problem of the planar convex hull. The special interest to this problem is caused by its simplicity which allows to investigate exemplary certain aspects. On the other hand, this interest is also caused by different applications.

Generally, the problem is given as following.

Problem R:

Given a set P of n points from R^2, the two-dimensional Euclidean space, find its convex hull as enumeration of the engaged points in clockwise order.

Theorem 1:

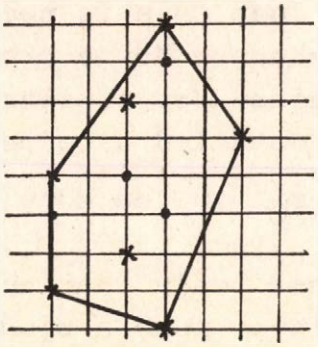
Problem R can be solved by O(nlogn) comparisons and O(n)

multiplications. Algorithms solving problem R with this effort are time optimal in the model of binary decision trees, quadratic resp. linear decision trees and in the model of algebraic computation trees, too.
These are well known results which can be found in [1].
In the following, we will investigate a restriction of problem R. We will pay special interest to three aspects:
- Is the restricted problem relevant to applications? (Section 2)
- What is the complexity of algorithms solving the general problem under the restriction? Are there better algorithms? (Section 2)
- Are lower bounds known for the general problem valid under the restriction? (Section 3 - 6)

2. A Special Convex Hull Problem

Problem $Z(m_1,m_2)$:
Given a set P of n points from $G = [0,m_1) \times [0,m_2) \cap Z^2$, in which m_1, m_2 are positive integers depending on n, find the convex hull of P as enumeration of the engaged points in clockwise order. $m_1 m_2 \geq n$.
The condition $m_1 m_2 \geq n$ ensures that all n points can be placed in the grid G.
This problem means all cases of application in which the input data are significantly discrete, i.e. the input data are integer multiples of a constant like e.g. the layout constant or the scanning constant of a camera. The field of view has finite dimensions. According with this, fields of application are the layout verification and the processing of digital images. Compare this approach with those of Karlsson [2] who refers to the same field of application.
At first, we give an upper bound, i.e. we sketch a scan line algorithm solving $Z(m_1,m_2)$ and determine its complexity. We qualify such algorithms as scan line algorithms which consist of at least two passes. The first pass sorts the input set with regard to a given ordering. The suceeding pass processes the input set in this order. Now, there are a lot of scan line algorithms solving problem R, and in which the first pass sorts the input with regard to the lexicographic order.
The second pass determines the convex hull by stepwise processing the input in this order. If we take into considera-

tion that only y-extreme points can take part in the convex hull, we can reduce the first pass to the determination of these points. $p = (x,y) \in P$ is a y-extreme point of P iff either $y \leq y'$ for all $p' = (x',y') \in P$ (p is y-minimum) or $y \geq y'$ for all $p' = (x',y') \in P$ (p is y-maximum). The convex hull is marked by "——", the y-extreme points are marked by "x" in the illustration.



The first pass determines all these points in lexicographic order. Their number is denoted by m_y. The second pass needs $O(1)$ multiplications and comparisons per point, by this $O(m_y)$ multiplications and comparisons at all. The total costs are $O(n\log m_y)$ comparisons in the first pass and $O(m_y)$ multiplications of an x- and y-coordinate and $O(m_y)$ comparisons in the second pass. We call the algorithm sketched above HULL. Because m_y can be $\Theta(n)$ in the case of problem R, HULL possesses a worst case behaviour of $\Theta(n\log n)$ comparisons and $\Theta(n)$ multiplications. However, we obtain $m_y = O(\min(n, 2m_1))$ in the case of problem $Z(m_1,m_2)$. Furthermore, multiplication on the grid $G = [0,m_1) \times [0,m_2) \cap Z^2$ can be done by $O(\log m_1 m_2)$ comparisons. Therefore, HULL needs $O(n\log m_y + m_y \log m_1 m_2)$ comparisons of coordinates to solve $Z(m_1,m_2)$.

For simplicity, we suppose in the following

$$m_1 \leq m_2 \qquad (1)$$

$$m_2 = O(n^{\alpha}) \text{ with } \alpha > 0 \qquad (2)$$

$$m_1 \leq n \qquad (3)$$

Condition (1) is rather of technical importance, i.e. it simplifies the following explanations. With condition (2) and (3) we try to choose such a domain of the parameters $m_1(n)$ and $m_2(n)$ which is of interest, i.e. in which problem $Z(m_1,m_2)$ differs essentially from problem R. Intuitively, we mean by this "well filled" grids. Furthermore, condition (1), (2), (3) make the two terms $n\log m_y$ and $m_y \log m_1 m_2$ of the worst case complexity of HULL comparable.

Theorem 2:
Under the condition (1), (2), (3) the algorithm HULL solves problem $Z(m_1,m_2)$ with $\Theta(n\log m_1)$ coordinate comparisons in the worst case.
In the following, we search for lower bounds of $Z(m_1,m_2)$. For this end, we have to define the model of computation, in which we investigate $Z(m_1,m_2)$. That this is necessary, will become clear, if we compare the upper bound provided by HULL with those of Karlsson [2] gained for similar problems, but in a more special model. For a detailed discussion we refer to [3].

3. Lower Bounds for $Z(m_1,m_2)$ in the Model of Binary Decision Trees

In this model we suppose only that any inner node possesses two edges. At any inner node a binary decision has to be made. The leaves contain the answers. Therefore, a tree correctly answering a question possesses at least so many leaves as there are different answers. The number of leaves is bounded by 2^h in which h is the hight of the tree. This well known argument provides us with the so-called information theoretic lower bound $h \geq \log m$ in which m denotes the number of distinct answers.
Here, we have to estimate the number m of distinct convex hulls. Two convex hulls are distinct iff they consist of different points or they consist of the same points, but the points are enumerated in different cyclic orders. h_{max} denotes the number of vertices in a convex hull for $Z(m_1,m_2)$ with maximally many vertices. Then,

$$m = \sum_{i=2}^{h_{max}} \binom{n}{i} (i-1)! \qquad (n \geq 2)$$

The determination of h_{max} is a deep problem. For this problem pay also attention to [3].
Obviously, $h_{max} \leq \min(n, 2m_1, 2m_2) = \min(n, 2m_1)$ by condition (1). But, h_{max} does not attain this upper bound in any case. This depends on the relation of m_1, m_2. Lemma A gives insight into this question, but it cannot give a complete answer.

Lemma A:
a) A convex hull with exactly $2m_1$ elements exists only for

$$m_2 \geq 2 + \left(\left\lfloor \frac{m_1+1}{2} \right\rfloor - 2 \right) \left(\left\lfloor \frac{m_1+1}{2} \right\rfloor - 1 \right) = \frac{m_1^2}{2} + O(m_1)$$

and $2m_1 \leq n$.

b) There exists a convex hull with $\Theta(m_1)$ many points for $m_1^2 = O(m_2)$.

c) There exists a convex hull with $\Theta(\sqrt{m_2})$ many points for $\sqrt{m_2} = O(m_1)$.

If we replace the condition $m_1 m_2 \geq n$ from the problem definition of $Z(m_1, m_2)$ by

$$\exists \varepsilon > 0 \; \exists n_0 \; \forall n \geq n_0 \quad m_1 m_2 (1-\varepsilon) \geq n \qquad (4)$$

then we can show the following theorem by means of lemma A. (Condition (4) means that the number of empty places in the grid G is at least proportional to n.)

Theorem 2a:

The algorithm HULL is time optimal for $Z(m_1, m_2)$ in the model of binary decision trees under condition (1)-(4) and (a) or (1)-(4) and (b).

(a) $m_1 = \Theta(n)$ and $m_1^2 = O(m_2)$

(b) $\sqrt{m_2} = \Theta(m)$ and $\sqrt{m_2} = O(m_1)$.

HULL is not time optimal in the same model under condition (1)-(4) and

(c) $m_1 = o(n)$.

By this theorem one can see that there exists a gap between the upper bound $O(m_1 \log n)$ of the information theoretic lower bound and the worst case behaviour of HULL, e.g. in the case $m_1 = o(n)$. This gap can only be closed when using more special computational models.

4. $Z(m_1, m_2)$ in Other Known Computational Models

Models of this kind are the model of algebraic computation trees and the model of quadratic resp. linear decision trees [1]. One sees at once that HULL solves $Z(m_1, m_2)$ in any of these three models. Recall the well known result from the literature that problem R, i.e. the general problem is not solvable in the model of linear decision trees.

Lower bounds are generally obtained in the models mentioned above by determining the number of different connected components of the input space. The curves bounding the components are linear, quadratic resp. algebraic depending on the chosen model. However, the decisive fact is the continuity of the input space. At most, the input space is R^n. All problems investigated in these models up to now possess

this property. Compare e.g. with [1]. The discrete nature of $Z(m_1,m_2)$ makes such an approach apparently difficult or even impossible. That's why we looked for a model of computation which reflects the nature of $Z(m_1,m_2)$ in a more adequate way. The finiteness of grid G suggests to choose coordinate comparisons as basic unit of the complexity measure.

5. A Special Model - the Model of $CC(m_1,m_2)$-trees

For our interest is directed towards geometric problems on planar point sets we presume the input to be in form of n coordinate pairs $r_1 = (x_1,y_1),\ldots,r_n = (x_n,y_n)$.

Definition:

A coordinate comparison tree, shortly CC-tree is a binary decision tree parameterisized by n in which the comparison at node v is of form

1) $x_{i_v} \geq x_{j_v}$, shortly x-comparison
2) $y_{i_v} \geq y_{j_v}$, shortly y-comparison
3) $x_{i_v} \geq c_v$ resp. $x_{i_v} \leq c_v$, shortly xc-comparison or
4) $y_{i_v} \geq d_v$ resp. $y_{i_v} \leq d_v$, shortly yd-comparison or
5) $x_{i_v} \geq y_{j_v}$ resp. $x_{i_v} \leq y_{j_v}$, shortly xy-comparison

with $i_v, j_v \in \{1,\ldots,n\}$ and c_v, $d_v \in R$.

A $CC(m_1,m_2)$-tree with $m_1(n),m_2(n)$ functions over Z^+ and with values from Z^+ is a CC-tree parameterisized by m_1 and m_2 in which it holds for c_v from 3) and d_v from 4) of the previous definition $c_v \in [0,m_1) \cap Z$ and $d_v \in [0,m_2) \cap Z$.

Assertions classifying the CC-model into the hierarchy of the well known computational models can be found in [3]. The model $CC(m_1,m_2)$ is exactly such powerful as the model of linear decision trees, but less efficient on inputs from $G = [0,m_1) \times [0,m_2) \cap Z^2$.

6. Lower Bounds for $Z(m_1,m_2)$ in the Model of $CC(m_1,_2)$-trees

The following theorem contains the main result.

Theorem 4:

The asymptotic lower bound $\Theta(n\log m_1)$ for problem $Z(m_1,m_2)$ holds in the model of $CC(m_1,m_2)$-trees under the following conditions:

$$\lim_{n\to\infty} m_1(n) = \infty \qquad (5)$$

$$\exists\, \varepsilon > 0\ \exists\, \varepsilon_2 > 0 \quad n(1+\varepsilon) < m_2(1-\varepsilon_2) \text{ , if n large enough} \qquad (6)$$

$$m_1(n) = o(n). \qquad (7)$$

Condition (5) is a rather technical condition. It seems to be

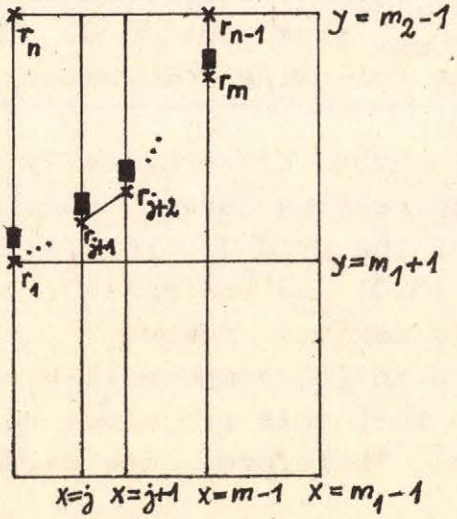
not too strong. Condition (6) and (7) are crucial for the proof. They can be summarized to the following: One coordinate (here the x-coordinate) grows slower than n. The other coordinate (here the y-coordinate) grows at least such fast as n. Observe that condition (7) is stronger than condition (3).

Sketch of Proof:

At first, we define special point configurations being inputs of $Z(m_1,m_2)$ which are called admissible, in the following way. m+2 points form the convex hull, in which

$m := \min(m_1, \lfloor \sqrt{2\varepsilon_2 m_2} \rfloor)$.

These points are marked by "x". The remaining points are equally distributed in the first m columns directly above the convex curve formed by $r_1,..,r_m$. They are marked by blocks (▮). As an important observation we mention the fact that these blocks don't overlap each other.



At first, we have to show that admissible configurations are input of $Z(m_1,m_2)$. This can be done by means of condition (5)-(7). Admissible point configurations are called partially monotonous or shortly pma-configurations if the indices of the points in each block ordered by increasing y-values form a monotonously increasing sequence.

Now, one can show the following lemma.

Lemma B:

Distinct pma-configurations lead to distinct leaves in any $CC(m_1,m_2)$-tree correctly solving $Z(m_1,m_2)$.

This is the main point of the proof. The remaining part of the proof is canonical.

Lemma C:

There are at least $(m!)^{\lfloor \frac{n}{m} \rfloor - 1}$ distinct pma-configurations.

Lemma D:

$$\log\left((m!)^{\lfloor \frac{n}{m} \rfloor - 1} \right) = \Theta(n \log m_1)$$

By these lemmas, the assertion of theorem 4 follows. From theorem 4 and theorem 2 follows at once

Theorem 2a:

Algorithm HULL solves problem $Z(m_1, m_2)$ time optimally in the model of $CC(m_1, m_2)$-trees under condition (5),(6),(7) and (2). Detailed proofs and related problems of computational geometry can be found in [3].

7. Final Comments

Theorem 2a sums up the main result. We tried to generalize this result in different ways. But up to now, these efforts remained unsucessful. Another direction of our interest was the determination of h_{max} from section 3. This problem has a close connection to the following combinatorial problem:

Problem:

Determine the maximal number of vertices lying on a concave , strongly monotonic increasing curve, consisting of finite lines between points of the grid $G = [0,k_1) \times [0,k_2) \cap Z^2$ with $k_1, k_2 \in Z$ starting in (0,0) and ending in (k_1-1, k_2-1).
$l(k_1, k_2)$ denotes this maximal number.
A PASCAL-program given in [3] computes $l(k_1, k_2)$ by recursion. It can be easily seen that this procedure reflects a problem of dynamic programing. Therefore, its direct computation is very expansive.
Because the author is not very experencied in the field of modern combinatorics we don't know whether there are results on this problem.

References

[1] F.P. Preparata and M.I. Shamos, Computational Geometry (Springer, New York, 1988)
[2] R.G. Karlsson, Scanline Algorithms on a Grid, Research Report LITH-IDA-86-30, University of Linköping, Dep. of Comp. Science and Inf. Science, 1986
[3] K. Unger, The Convex Hull Problem on grids, Report, K.-Weierstraß-Institut Berlin, 1988

IMYCS'88, Smolenice Castle, November 14-18, 1988.

ON OPTIMAL REALIZATION OF ALGORITHMS ON PIPELINED FUNCTIONAL UNITS

A.V. VOEVODIN

Institute of Problems of Cybernetics, Moscow, USSR

In the paper the following problems are discussed: the problem of time optimal and efficient realization of a set of informationally independent algorithms on a pipelined functional unit (f.u.) system and the problem of f.u. number reducing in the system at the expense of more detail pipelinization of units without significant increase of realization time for arbitrary algorithms.

Let us consider a set Ω of basic operations of $s \geqslant 1$ types. The basic operation of i-th type is defined by function from X^{α_i} to X^{β_i}, where X is space of words and α_i, β_i are naturals, $i=\overline{1,s}$. Let us consider a set $\{A\}$ of computational algorithms A, which can be presented by oriented graphs without circuits. Every graph vertex corresponds to some basic operation from Ω, the arcs

express the precede conditions between the algorithm operations.

Let us realise the algorithms on f.u. system, which consists of N f.u. of s types, $N_i \geq 1$ units of i-th type, $i=\overline{1,s}$, $N_1+\ldots+N_s=N$. The f.u. of i-th type has α_i inputs, β_i outputs, $k_i \geq 1$ stages and is used for realization of basic operations of i-th type from Ω. All f.u. to be synchronously functioning under the same clock periods τ. Before the every next clock periods every f.u. input may be connected with every f.u. output. Operands for f.u. inputs may be also obtained from system's storage. The results are transmitted either on f.u. inputs or in storage, where they are stored for usage in future.

The realization of algorithm A on f.u. system will be presented by scheduling. The scheduling P is a timetable with N rows and t columns. The rows of timetable correspond to f.u. and columns - to f.u. system clock periods. Not more than one vertex of algorithm graph can be located according to defined rules in the timetable's cell. The corresponding to i-th type operations vertexes are located in the rows corresponding to f.u. of i-th type. If the vertex v, corresponding to i-th type operations, is located in j_v-th column and precedes the vertex u from j_u-th column, then $j_u \geq j_v+k_i$. The execution of i-th type operation from j-th column of scheduling begins in the beginning of j-th clock period. The results of operation are being available for usage in the beginning of $(j+k_i)$-th clock period, $t \geq j+k_i-1$.

The main criteria of optimal scheduling are: time t and

average efficiency G. The algorithm A realization time on f.u. system according to scheduling P, expressed in clock periods, equals to the number of columns t. The average efficiency of f.u. system to be defined according to the formula:

$$G = \sum_{j=1}^{H} G_j/H, \qquad \text{where} \quad H = \sum_{i=1}^{s} k_i N_i ,$$

as a sum of efficiencies G_j of all H stages of the f.u. system, devided by H. The efficiency G_j of j-th stage is defined as ratio of clock periods number for the stage processing time to the total clock periods number for f.u. system processing time t obtained according to the scheduling P. The formula for efficiency can be written in the following form:

$$\dot{G} = \Big(\sum_{i=1}^{s} k_i m_i \Big) \Big/ \Big(t \sum_{i=1}^{s} k_i N_i \Big) ,$$

where m_i - a number of i-th type operations in algorithm A. It is obvious that the product Gt do not depend on algorithm A realization scheduling of f.u. system. That's why the problem of time optimal scheduling for the algorithm A and the f.u. system is equivalent to the problem with efficiency criterion. The optimal scheduling sets are identical.

The time optimal scheduling problem for arbitrary algorithm A and f.u. system is NP-hard [2]. Let us note $t_{min} = \min_{\{P\}} t$ the minimal realization time of algorithm on f.u. system. Very simple polynomial time scheduling algorithm which for arbitrary computational algorithm A gives the estima-

tion $t/t_{min} < s+1$ can be easily found. On the other hand polynomial time scheduling algorithm for arbitrary computational algorithm A and any f.u. system, such as $t/t_{min} < s$ for $s \geqslant 2$, hasn't been found yet.

One of possible approaches to the optimal scheduling problem is the research of practically used computational algorithm's structure. As the matter of fact in the large-scale computational algorithms independent and often identical parts A_j can be found. Let us say that the algorithms A_j are independent if their graphs are not connected one with the others. For the set of n independent and generally speaking different algorithms A_j, $j=\overline{1,n}$, simple and convenient in usage necessary and sufficient conditions of scheduling existance and scheduling algorithms with efficiency asymptotically equal to 1 can be found. Let for every independent algorithm A_j, $j=\overline{1,n}$, of arbitrary structure the condition

$$0 < \frac{m_1^j}{N_1} = \ldots = \frac{m_s^j}{N_s} \leqslant \gamma, \qquad (1)$$

where m_i^j - i-th type operations number in algorithm A_j, $j=\overline{1,n}$, is satisfied.

Theorem 1. An arbitrary set of n independent algorithms A_j under the condition (1) can be realized on the given f.u. system with the efficiency $G_n \to 1$ as $n \to \infty$.

Let's now consider arbitrary independent algorithms A_j sets, which satisfy the condition

$$1 \leqslant m_j \leqslant m, \qquad (2)$$

where m_j - the total number of operations in algorithm A_j,

$j=\overline{1,n}$.

Theorem 2. The set of independent algorithms A_j, which satisfy the condition (2), can be realized on the given f.u. system with efficiency $G_n \to 1$ as $n \to \infty$ iff

$$\frac{M_n^i}{M_n^1} \to \frac{N_i}{N_1} \quad \text{as} \quad n \to \infty ,$$

for $i=\overline{1,s}$, where M_n^i - the total i-th type operations number in the set of n algorithms A_j.

Theorem 3. The n-times computation of algorithm A_o on the given f.u. system for different independent initial data can be realized with efficiency $G_n \to 1$ as $n \to \infty$ iff A_o satisfies the condition (1).

Under the theorems proving polynomial time scheduling algorithms with $\lim_{n\to\infty} G_n=1$ are constructed. It is obvious that $t^n/t^n_{min} \to 1$ as $n \to \infty$, where t^n - scheduling time for the set of n algorithms.

In conclusion we are to present two lemmas on the comparison of two different f.u. system possibility. We are to mark the characteristics of the system by "old" and "new". Let's note the algorithm realization time as $\tilde{t}=t\cdot\tau$ (to be measured in seconds).

Lemma 1. If $\tau^{new}=\tau^{old}/r$, $k_i^{new}=k_i^{old}\cdot r$, k_i^{new} are integers, $N_i^{new}=\lceil N_i^{old}/r \rceil$, $r>1$, $i=\overline{1,s}$, then for every algorithm can be provided $\tilde{t}^{new} \leq R\tilde{t}^{old}$, where

$$R = 1+\frac{r-1}{rk_{min}^{old}} < 1+\frac{1}{k_{min}^{old}} \leq 2 , \qquad k_{min}^{old} = \min_{1\leq i\leq s} k_i^{old}.$$

Lemma 2. If $\tau^{new}=\tau^{old}\cdot r$, $k_i^{new}=k_i^{old}/r$, k_i^{new} are integers, $N_i^{new}=\lceil N_i^{old}\cdot r\rceil$, $r>1$, $i=\overline{1,s}$, then for every algorithm can be provided $\tilde{t}^{new}\leq\tilde{t}^{old}$.

Scheduling for system noted by "new" are constructed from scheduling for system noted by "old". The transformation algorithms have linear time complexity. We are to emphasize that f.u. number of the system noted by "new" in the condition of Lemma 1 is approximately 1/r-th of the f.u. number of the system noted by "old", but the realization time for arbitrary algorithm $A\in\{A\}$ and the system noted by "new" increase insignificantly.

REFERENCES

[1] M.R. Garey and D.S. Johnson, Computers and Intractability. A Guide to the Theory of NP-Completeness (Bell Lab., 1979).

[2] G.I, Marčuk and V.E. Kotov, Modul'naja asinhronnaja razvivaemaja sistema (Preprints N° 86, 87, Novosibirsk, VC SO AN SSSR, 1978).

[3] A.V. Voevodin, Nekotorye voprosy issledovanija konveĭernyh sistem s perestraevaemoĭ kommutaciei, in: Arhitektura ĖVM i čislennye metody (M., OVM AN SSSR, 1983) 53-74.

[4] A.V. Voevodin, Ob effektivnoĭ realizacii potoka vyčislitel'nyh algoritmov na funkcional'nyh ustroĭstvah konveĭernogo tipa, in: Programmnoe obespečenie vyčislitel'nyh kompleksov (M., MGU, 1985) 49-54.

[5] A.V. Voevodin, Modelirovanie vyčislitel'nyh processov v konveĭernyh sistemah, Diss. na soisk. učen. step. kand. fiz.-mat. nauk, M., MGU, 1983.

ALFAPRINT