tanulmányok

185/1986

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 4TH INTERNATIONAL MEETING OF YOUNG COMPUTER SCIENTISTS IMYCS'86

Smolenice Castle, Czechoslovakia, October 13-17, 1986

edited by:

J. DEMETROVICS

Computer and Automation Institute, Budapest

and

J. KELEMEN

Comenius University, Bratislava

Computer and Automation Institute of the
Hungarian Academy of Sciences, Budapest, 1986

Tanulmányok 185/1986
Studies 185/1986

Felelős kiadó:

KEVICZKY LÁSZLÓ

ISBN 963 311 214 1
ISSN 0324-2951

P R E F A C E

This volume contains the texts of talks presented at *The 4th International Meeting of Young Computer Scientists* held at Smolenice Castle, Czechoslovakia, October 13-17, 1986.

The Meeting was organized by the *Association of Slovak Mathematicans and Physicists* in cooperation with several other institutions and organizations. The aim of the Meeting was to promote research of beginners in Computer Science, to focus their professional attention to some distinguished problems, and to create an opportunity for establishing professional relations.

The Proceedings include the texts of invited lectures and the texts of 20 short communications selected from about 50 submissions by the Programme committee of IMYCS'86. It includes texts of (some) evening section talks, too. All the texts have been completed in camera-ready form by the authors.

We wish to express our gratitude to all of invited speakers, as well as to *E. Csuhaj-Varju* (Budapest, Hungary), *J. Dassow* (Magdeburg, GDR), *S.K. Dulin* (Moscow, USSR), *J. Pittl* (Prague, CSSR) and *M. Szijártó* (Győr, Hungary) for their active participation in the work of the Programme committee.

Special thanks go to *Alica Kelemenová* (Bratislava, ČSSR) for chairing the Programme committee, and to the *Computer and Automation Institute of the Hungarian Academy of Sciences* in Budapest for publishing the Proceedings.

Budapest and Bratislava, June 1986

János Demetrovics
Jozef Kelemen

TABLE OF CONTENTS

INVITED LECTURES

SHORT COMMUNICATIONS

EVENING SESSION CONTRIBUTIONS

INVITED LECTURES

Proc.IMYCS '86 October 13-17,1986
Smolenice Castle,ČSSR

LOWER BOUND TECHNIQUES FOR VLSI ALGORITHMS

J. Hromkovič

Department of Theoretical Cybernetics
Comenius University
842 15 Bratislava
Czechoslovakia

1. INTRODUCTION

The basic concept of complexity theory for VLSI was given by Thompson [32,33]. Since hundreds of papers dealing with VLSI algorithms were published in the last six years we have no chance to consider all of them. The aim of this paper is to outline an short overview involving the basic concepts in the proving of lower bounds for different complexity measures of VLSI algorithms the new approaches making the lower bound proof techniques more successful, and the use of the idea of "information transfer for VLSI" for the obtaining of lower bounds on different complexity measures of another computing models. In the case that the reader is interested in other questions concerning VLSI theory too, the monograph of Ullman [34] is much recommended.

First, let us give the definition of the notion "problem", and a short, informal definition of the notion "VLSI circuit".

Let $X = \{x_1,\ldots,x_n\}$, $Y = \{y_1,\ldots,y_m\}$ be sets of Boolean variables. A problem instance from the input variables X to the output variables Y is a set of Boolean functions $f_1,f_2,\ldots,f_m$ such that $f_i : X \to \{0,1\}$ and $y_i = f_i(x_1,x_2,\ldots,x_n)$ for $i = 1,\ldots,m$.

A problem is an infinite sequence of problem instances, where each two instances in the sequence have a different size parameter n.

We have no space to formally specify the notion "VLSI cir-

cuit". So, we give only an informal abstaction of this notion.

A VLSI graph can be wiewed as a directed graph embeded in the lattice with the following properties.

(1) the sum of output and input edges from any vertex is bounded by 4,

(2) each square of the lattice has one of the following contentses:

  (a) an vertex of the graph

  (b) one line going in the horisontal or in the vertical direction (this line is a part of an edge of the graph)

  (c) two crossing lines, one going in the horisontal direction, another in the vertical direction (this depicts the place of two crossing edges without any vertex of the embeding of the directed graph in the plane)

  (d) the empty contents.

The space complexity of an VLSI graph is the area of a minimal rectangle involving all non-empty squares of the lattice.

We can obtain a VLSI circuit of n input variables $x_1,\ldots,x_n$ and m output variables $y_1,\ldots,y_m$ from a VLSI graph in the following way:

(1) we assign to each vertex of the VLSI graph an processor which has the same number of Boolean inputs (outputs) as the indegree (outdegree) of the vertex is,

(2) a pair (v,t) is related to each input and output variable, where v is a vertex and t is a nonnegative number (for an input variable it means that it will be read by the VLSI circuit through the vertex in the t-th time unit of the computation; analougsly for an output variable).

The space complexity of an VLSI circuit is the space complexity of the corresponding VLSI graph.

The VLSI circuit computes in such a way that all processors are working in each time unit, and the information (a Boolean value) between two connected processors flows exactly one time unit. The time complexity of an VLSI circuit is $\max\{t \mid (v,t)$ is a pair related to an output variable$\}$.

We shall not specify what it means that "a VLSI circuit solves a problem instance" because we hope that it is clear from

the above introduced. Solutions to problems are sequences of circuits, one for each instance of the problem. So, the time (T) and area (A) complexities of a problem can be defined as functions from positive integers to positive integers in the obvious way (note that VLSI circuits are a nonuniform computing model).

The paper is divided in six sections. Section 2 and 3 resp. involves the outline of lower bound techniques for the complexity measure A and AT respectively. The technique for proving lower bounds on the most studied complexity measure AT^2 is presented in Section 3. The abstaction of this technique based on the notion "communication complexity" is introduced in this section too. Section 4 involves a new approach to defining the notions introduced in the previous section in the order to make the lower bound technique for AT^2 more successful. Section 6 consists of some examples showing that the notion "communication complexity" can be used to obtain the lower bounds on different complexity measures of other computing models.

2. LOWER BOUNDS ON THE AREA

The area of a VLSI circuit solving a specific problem was investigated in several papers (see, for example, [3,5,9,18,19]). The reason to deal with the area complexity measure follows from the technology. If we are able to produce a good, special chip of area complexity A with a probability p (for example, if p = 1/10 then it means that 10% of the produced chips are good) then the probability of producing the good chip of area complexity 2A is p^2 (1% in our example). So, the charge of the VLSI chips growths exponentially with the area complexity of these chips.

The obtaining of lower bounds on the area of VLSI circuits is based on the following fact. Each circuit having area A cannot remember more than A bits from one time unit to the next one. We can define, for each time unit, the state of a circuit as the sequence of the output bits of all processors in the circuit. Using this notion we shall formulate a general "algorithm" for proving lower bounds on area of VLSI circuits computing specific problems.

"A - algorithm" :

I n p u t : A problem instance P with input variables X and output variables Y.

S t e p 1 - Prove, for a $X_1 \subseteq X$ and $Y_1 \subseteq Y$, that there is a time unit t such that all input variables from X_1 have to be read before the time unit t, and all output variables from Y_1 have to be computed after the time unit t in any VLSI circuit solving P.

S t e p 2 - Prove, for a number d, that there are d different assignments of values to the input variables from X_1 which require distinct assignments of values to the output variables from Y_1.

O u t p u t : $A \geq \log_2 d$.

The correctness of "A-algorithm" follows from the fact that the VLSI circuit that is in the same state in the same time unit t for two different inputs (according to X_1) cannot distinguish between these two inputs, and has to compute the same values for all output variables computed after the time unit t.

Using "A-algorithm" one can prove, for example, that the sorting of m digits of the length $\lfloor \log_2 m \rfloor + 1$ in the binary coding requires $A \geq m$.

3. LOWER BOUNDS ON THE TRADEOFF AT

There is a very simple lower bound proof technique for the complexity measure AT considered, for example, in [4,6,24-26,32, 33]. It is based on the following theorem.

Theorem 2.1 Let P be a problem instance with input variables X, and output variables Y. Let $d = \max\{|X|, |Y|\}$. Then $AT \geq d$.

Proof. In each time unit the VLSI circuit can read (write) at most A bits.

Using Theorem 2.1 we have that $AT \geq m \log_2 m$ for any VLSI circuit sorting m numbers of the length $\log_2 m$.

4. LOWER BOUNDS ON AT^2 AND COMMUNICATION COMPLEXITY

The complexity measure AT^2 is the most studied area-time tradeoff in VLSI theory [1,2,4,9,11-13,15,23,30,32-34]. Opposite the lower bounds on A and AT based on memory requirements the lower bounds on AT^2 are based on the requirements on information flow within the chip.

In what follows we shall consider the circuits with 1/3-property, where 1/3 property means that no processor in the circuit is assign to more than 1/3 input variables. Clearly, in the case that a circuit has not 1/3-property it must satisfy $T \geq n/3$ (n is the number of input variables), what implies $T^2 = \Omega(n^2)$ ($AT^2 = \Omega(n^2)$).

To outline the "strategy" for proving lower bounds on AT^2 we need the following lemma.

Lemma 3.1 Let P be a problem instance with the set of input variables X, and let C be a circuit with 1/3 property solving this problem. Then there is a line involving at most one single jog (see Fig.1) that divides the circuit C into two parts, each having assigned between 1/3 and 2/3 input variables.

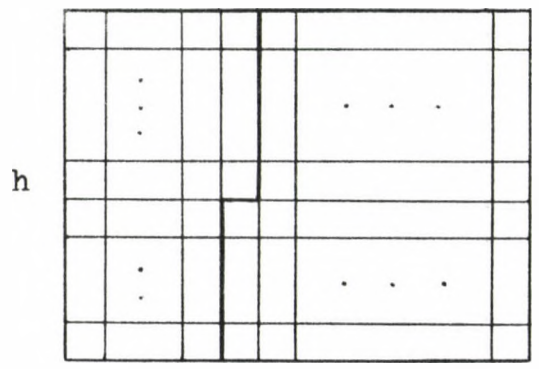


Fig.1

Now, the idea consists in proving that at least d bits must flow through the line depicted at Fig.1, for a positive integer d. Then assuming $k \geq h$ we have $hT \geq d$, i.e. $AT^2 \geq d^2$.

Let P_n be a problem instance with input variables X, $|X| \geq n$, and output variables Y. A partition for P_n is a division of X into two disjoint sets X_L and X_R, $X_L \cup X_R = X$, $1/3 \leq |X_L|, |X_R| \leq 2/3$, and a division of Y into Y_L and Y_R, $Y_L \cup Y_R = Y$. Obviously, a partition can be assign to each line dividing a circuit into two parts. Now, let us formulate an "algorithm" for proving lower bounds on AT^2.

"AT^2 - algorithm"

I n p u t : A problem instance P_n.

S t e p 1. Let $\pi_1, \pi_2, \ldots, \pi_k$ be all different partitions for P_n, and let $\pi_i = (X_L^i, X_R^i, Y_L^i, Y_R^i)$. Find, for each $i \in \{1,2,\ldots,k\}$, the largest set A_i of input assignments such that any two assignments of input variables in A_i require to be distinguished by the information flowing across the boundary.

S t e p 2. Compute $d = \min\{|A_i| \mid i = 1,\ldots,k\}$.

O u t p u t : $AT^2 \geq (\log_2 d)^2$.

The introduced "AT^2-algorithm" based on so called "crossing sequences" or "fooling sets" is correct because if a circuit has the same information flow through the line for two inputs α and β then the input constructed from $\alpha_R\beta_L$ (α_R is the part of input assignment α restricted by X_R, analougsly β_L) must have the same output values in Y_L (Y_R) as β (α).

The abstraction of this concept led to the definition of the notion "communication complexity" for language recognition in [23]. Let us give an informal definition of this complexity measure.

Suppose that a language $L \subseteq \{0,1\}^*$ must be recognized by two distant computers. Each computer receives aproximately half of the input bits, and the computation proceeds using some protocols for communication between the two computers. The minimum number of bits that has to be exchanged in order to successfully recognize $L \cap \{0,1\}^n$, minimized over all partitions of the input bits into two aproximately equal parts, and considered as a functions of n, is called the communication complexity of L.

The communication complexity C of L provides a direct lower bound $AT^2 \geq C^2$ on any circuit recognizing L. This complexity measure was studied in several papers [2,9-13,15,17,23], where the basic results concerning the strong hierarchy of communication complexity, relation between determinism and nondeterminism in communication complexity model, closure properties of the classes of languages determined by communication complexity, the relation between Chomsky hierarchy and communication complexity hierarchy, lower bounds on the communication complexity of specific languages, and the properties of special types of communi-

cation complexity model were established.

5. AN IMROVED LOWER BOUND ARGUMENT FOR AT^2.

The technique introduced in the previous section has the following two lacks:

1, There are very hard problems according to AT^2 which can be solved with small "information transfer".
2, It is very hard to prove a high lower bound on "information transfer" as the minimum over all partitions. It holds in the cases too, where it seems to be obvious that a high communication complexity is required.

We shall try to show the background of these lacks. Let us have a problem consisting of a constant number of subproblems with disjoint input variables. Let some of these subproblems require linear (maximal) communication complexity, and let a small (constant, for example) communication complexity suffices to obtain the solution of the problem in the case that the solutions of the subproblems are known. Then, if we take a partition of the input bits which gives the input bits of some subproblems to the first computer, and the input bits of additional subproblems to the second computer, the problem can be solved with small (constant) communication complexity. On the other hand the solution of the problem can require $AT^2 = \Omega(n^2)$. In [12] it is shown how a problem with zero communication complexity can be constructed from a problem with linear communication complexity without decreasing the complexity AT^2. In [10] it is shown that a Boolean function with linear communication complexity can be obtained as a disjunction of two Boolean functions with constant communication complexity. Clearly, this cannot be true for AT^2.

Now, let us consider the second lack. Let P_k be a problem instance depending on two sets of input variables X, and Y, but in another way on X as on Y. There are cases that, for partitions that divide the input bits such that X is divided on two aproximately equal-sided parts, the proof of the lower bound on information transfer is not very hard. But, for additional partitions

the proof can be much harder.

This led to new approaches to the defining of the notions "fooling sets" and "communication complexity" in [2,15]. They are based on the fact that we need not to take minimum over all partitions. In fact we can choose any subset Z of input variables and take minimum over all partitions of the input variables X dividing Z into two aproximately equal-sided parts only. So, we can improve the "AT^2-algorithm" by adding the initial step: "Choose a suitable $Z \subseteq X$", and following the original algorithm for partitions with 1/3-property according to Z.

How this new approach helps for the use of communication complexity is shown in [15]. For example, a specific language with $AT^2 = \Omega(n^2)$ having constant original communication complexity, and linear new communication complexity is constructed, and it is proved that almost all languages having sublinear original communication complexity require in fact linear communication complexity.

6. COMMUNICATION COMPLEXITY AND LOWER BOUNDS FOR OTHER COMPLEXITY MEASURES

We shall show in this section how the idea based on the notion "communication complexity" was used to obtain lower bounds for distinct computing models.

6.1 Area and space complexity of Boolean circuits

The problem of determining lower bound on area and space needed for the computation of problems on the well-known computing model - Boolean circuit was studied in [31], where the lower bound $\Omega(n^{3/2})$ for a specific Boolean function was established. Using a special type of communication complexity the strongest lower bound $\Omega(n^2)$ and $\Omega(n^{3/2})$ resp. was obtained for area and space complexity respectively in [21].

6.2 Linear lower bounds on the number of gates of CA-circuits

The CA-circuits introduced in [14] are a generalisation of unbounded fan-in Boolean circuits. Defining the communication complexity of CA-circuits the method for obtaining linear lower

bounds on the number of gates was developed in [14].

6.3 Lower bounds on branching programs

The branching programs were introduced in [22,28] as a tool for obtaining lower bounds on the space complexity of sequential algorithms. A special type of communication complexity was used to obtain new lower bounds for branching programs [27]. In some sence the communication complexity is related to the width of branching programs.

6.4 Information flow among distinct processes in distributed computing

The communication complexity model can be used to study the requirements on the information flow among distinct processes in distributed computing [8,16]. The model is considered too if the question, whether the computation facilities of communication computer in a computer network with a special topology can decrease the amount of submitted data, is investigated.

ACKNOWLEDGEMENT

I would like to thank **Erika Kupková** who calls my attention to the paper of Yao, Ullman and Yanakakis [2], where the first ideas concerning the new approach to defining the notion "information transfer" were presented, what enables me to compare the ideas of [2] with my effort to improve the specification of the notion of communication complexity in [15].

REFERENCES

1. **Abelson,H.**: Lower bounds on information transfer in distributed computations. Proc. 19th Annual **IEEE FOCS**, pp.151-158,1978
2. **Aho,A.V. - Ullman,J.D. - Yannakakis,M.**: On notions of information transfer in VLSI circuits. Proc. 15th **ACM STOC**, pp. 133-139, 1983
3. **Baudet,G.M.**: On the area required by VLSI circuits. In: Kung, Sproul, and Steele 1981, pp.100-107
4. **Brent,R.P. - Goldschlager,L.M.**: Some area time tradeoffs for VLSI. **SIAM J. Comput.** 11, No.4, 1982, pp.737-747.

5. Brent,R.P. - Kung,H.T.: The chip complexity of binary arithmetic. Proc. 12th Annual ACM STOC, pp.190-200, 1980.

6. Brent,R.P. - Kung,H.T.: The area - time complexity of binary multiplication. J. ACM 28, No.3, 1981, pp.521-534.

7. Chandra,A.K. - Furst,M.L. - Lipton,R.J.: Multiparty protocols. Proc. 15th Annual ACM STOC, 1983, pp.94-99.

8. Ďuriš,P.: personal communication.

9. Ďuriš,P. - Galil,Z. - Schnitger,G.: Lower bounds on communication complexity. Proc. 15th Annual ACM STOC, pp.81-91, 1984.

9. Ďuriš,P. - Sýkora,D. - Vrťo,I. - Thompson,C.D.: Tight chip area lower bounds for discrete Fourier and Walsh-Hadamard transformations. Infor. Proces. Let. 21, 1985, 245-247.

10. Gubáš,X. - Vaczulík,J.: Closure properties of the families of languages defined by communication complexity. ŠVOČ 1986-section Theoretical Cybernetics and Mathematical Informatics, Comenius University, Bratislava 1986 (in Slovak).

11. Hromkovič,J.: Communication complexity. Proc. 11th ICALP, Lect. Notes in Computer Science 172, Springer-Verlag 1984, pp.235-246.

12. Hromkovič,J.: Relation between Chomsky hierarchy and communication complexity hierarchy. Acta Mathematica Universitates Comenianae 1986, to appear.

13. Hromkovič,J.: Normed protocol and communication complexity. Computers and Artificial Intelligence 3, No.5, 1984, 415-422.

14. Hromkovič,J.: Linear lower bounds on unbounded fan-in Boolean circuits. Infor. Proces. Let. 21, 1985, 71-74.

15. Hromkovič,J.: A new approach to defining the communication complexity for VLSI. Proc. 12th MFCS 86, Lect. Notes in Computer Science, Springer-Verlag, to appear.

16. Ja´Ja,J. - Prasanna Kumar,V.K. - Simon,J.: Information transfer under different sets of protocols. SIAM J. Comput. 13, No.4 (1984), pp.840-849.

17. Kurcabová,V.: Communication complexity. Master thesis, Dept. of Theoretical Cybernetics, Comenius University, Bratislava 1985 (in Slovak).

18. Leiserson,C.E.: Area efficient graph algorithms (for VLSI). Proc. 21st Annual IEEE FOCS, pp.270-281, 1980.

19. Leiserson,C.E.: Area efficient VLSI computation. MIT Press, Cambridge 1983, Mass.

20. Lipton,R.J. - Sedgewick,R.: Lower bounds for VLSI. Proc. 13th Annual ACM STOC, pp.300-307, 1981.

21. Ložkin,C.A. - Rybko,A.N. - Sapoženko,A.A. - Škalikova,N.A.: An approach to obtaining lower bounds on space of Boolean circuits. Banach Centre publ. (in Russian), to appear.

22. Masek,W.: A fast algorithm for the string editing problem and decision graph complexity. M.Sc. thesis, MIT, May 1976.

23. Papadimitriou,C.H. - Sipser,M.: Communication complexity. J. of Computer and System Sciences 28, 1984, pp.260-269.

24. Preparata,F.P.: A mesh-connected area-time optimal VLSI integer multiplier. In Kung, Sproull, and Steele 1981, pp. 311-316.

25. Preparata,F.P. - Vuillemin,J.E.: Area-time optimal VLSI networks for multiplying matrices. Infor. Proes. Let. 11, No.2, 1980, pp.77-80.

26. Preparata,F.P. - Vuillemin,J.E.: Area-time optimal VLSI networks for computing integer multiplication and discrete Fourier transformation. Proc. 8th ICALP 81, Lect. Notes in Computer Science 115, Springer-Verlag 1981, pp.29-40.

27. Pudlák,P.: personal communication.

28. Pudlák,P. - Žák,S.: Space complexity of computations. Unpublished manuscript, 1982.

29. Savage,J.E.: Planar circuit complexity and the performance of VLSI algorithms. In Kung, Sproul, and Steele, 1981, pp. 61-67.

30. Savage,J.E.: Area-time tradeoffs for matrix multiplication and related problems in VLSI models. J. Computer and System Sciences 20, No.3, pp.230-242.

31. Škalikova,N.A.: Kletočnyje avtomaty. Ph.D.thesis, Dept. of Mathematical Cybernetics, Moscow State University 1979 (in Russian).

32. Thompson,C.D.: Area-time complexity for VLSI. Proc. 11th Annual ACM STOC, 1979, pp.81-88.

33. Thompson,C.D.: A complexity theory for VLSI. Ph.D.thesis, Carnegie - Mellon Univ., Pittsburg, Pa.

34. Ullman,J.D.: Computational Aspects of VLSI. Computer Science Press 1984, 495p.

35. Yao,A.C.: Some complexity questions related to distributed computing. Proc. 11th Annual ACM STOC, pp.209-213, 1979.

36. Yao,A.C.: The entropic limitations of VLSI computations. Proc. 13th Annual ACM STOC, pp.308-311, 1981.

Proc.IMYCS '86 October 13-17,1986
Smolenice Castle,ČSSR

THE EQUIVALENCE OF MAPPINGS ON LANGUAGES

Juhani Karhumäki

Department of Mathematics
University of Turku
20500 Turku, Finland

Abstract. We define the notion of the equivalence of mappings on languages in three different ways and call them universal equivalence, existential equivalence and equivalence with multiplicities. We survey recent results on this topic, as well as state some open problems.

1. Introduction

Since the beginning of the automata theory one of the most natural problems of the field has been the equivalence problem for automata or other devices of a certain type, that is to say, the problem of finding an algorithm or proving the nonexistence of such to decide whether two given automata behave in the same way, or in other words, are equivalent. Our intention here is to point out that the research in this problem area is still now - 30 years later - quite active and that there are many attractive unanswered problems left.

We formulate our basic problem as follows. Let L be a family of languages over a finite alphabet Σ and θ a family of (not necessarily single-valued) partial mappings or devices defining such from the free monoid Σ^* into another free monoid. Then we want to decide whether, for a given L from L and two mappings from θ, these mappings are "equivalent" on L. This problem in connection with morphisms, i.e., the "morphic equivalence for languages", was introduced by Culik and Salomaa in

[CS], which was a starting point for quite an active research.

Clearly, the above formulation includes the problem of deciding the equivalence of two automata with outputs, i.e., the equivalence problem for finite transducers. Other typical cases we shall be dealing with are the cases when L is the family of regular languages and θ is either a morphic mapping, i.e., a composition of morphisms and inverse morphisms, or a finite substitution. In particular, we shall be looking for borderlines between the decidable and undecidable equivalence problems in these cases.

If the partial mappings σ and τ are many-valued, in other words nondeterministic, then there are (at least) three different possibilities to define the notion of the equivalence of σ and τ on L. In each case the equivalence is word-by-word equivalence, which means that the mappings must behave in a similar way on each of the words of L. The most natural definition of the equivalence is the ordinary one which we refer to as the universal equivalence: σ and τ are universally equivalent on a word x if $\sigma(x)$ and $\tau(x)$ coincide as sets. They are existentially equivalent on x if either $\sigma(x)$ and $\tau(x)$ have a nonempty intersection or both are empty, and they are equivalent with multiplicities if $\sigma(x)$ and $\tau(x)$ are the same as multisets.

The rest of this paper is organized as follows.

After introducing the problems in details in Section 2 we recall in Section 3 the main decidability and undecidability results concerning the (universal) equivalence problem for transducers. We consider both one-way and two-way finite transducers, but do not deal with more general devices. Some of the results are rather new and are obtained by using techniques which allow to test the equivalence not only on the domains of the transducers but also on a given HDTOL language.

In Section 4 we consider the universal equivalence problem for different kinds of morphic and related mappings on regular languages. We are able to detect a sharp borderline between the decidability and the undecidability.

Finally, in Section 5 we discuss two other types of equivalences, although we have only a few results in this direction.

We conclude this paper by giving a couple of open problems which, we believe, are quite interesting and important.

As a survey this paper does not contain any essentially new results. Neither are the proofs given, only a few outlines or simple constructions are shown. However, the references to complete works are always mentioned.

2. Preliminaries and the problems

We assume that the reader is familiar with the basics of formal language theory, cf.[H] or [Be]. Consequently, we recall here only very few definitions.

According to [Be] we denote a finite one-way transducer by a sixtuple $T = \langle \Sigma, \Delta, Q, q_0, F, E \rangle$, where Σ and Δ are the input and output alphabets respectively, Q is the set of states, q_0 is the initial state, F is the set of final states and $E \subseteq Q \times \Sigma^* \times \Delta^* \times Q$ is the set of transitions. The relation realized by T is denoted by $|T|$ and it can be viewed as a partial many-valued mapping from Σ^* into Δ^*. Forgetting the output structure of T we obtain the underlying finite (generalized) automaton of T.

Clearly, a finite transducer may produce an infinite number of outputs for a given input. However, in many cases we want to consider only the following restricted classes of transducers: (i) T is <u>k-valued</u>, for some given $k \geq 1$, if for each input word there exists at most k different output words, (ii) T is <u>k-ambiguous</u>, for some given $k \geq 1$, if for each input word there exists at most k different accepting computations, and (iii) T is <u>deterministic</u> if $E \subseteq Q \times \Sigma \times \Delta^* \times Q$ and for each $q \in Q$ and $a \in \Sigma$ the cardinality of the set $(\{q\} \times \{a\} \times \Delta^* \times Q) \cap E$ is at most one. Further a transducer is <u>finite-valued</u> (resp. <u>finite-ambiguous</u>) if it is k-valued (resp. k-ambiguous) for some $k \geq 1$. Observe also that our deterministic transducers are often called deterministic gsm's or deterministic sequential transducers. (Indeed, T is a gsm if $E \subseteq Q \times \Sigma \times \Delta^* \times Q$).

All the above restrictions can be defined in a natural way in connection with two-way finite transducers as well, cf. [EY].

Next we fix our notation for some families of partial mappings. We assume that the domain and range alphabets are fixed, say Σ and Δ. We denote by H and S the families of morphisms and finite substitutions, respectively. Clearly, each partial many-valued mapping σ considered as a relation has the inverse and thus defines the unique such mapping, the inverse of σ which is denoted by σ^{-1}. Similarly, any composition of such mappings defines the unique such mapping. Consequently, if θ_1 and θ_2 are families of partial many-valued mappings so are θ_1^{-1} and $\theta_1 \circ \theta_2$ (where we first apply mappings from θ_2). In particular, H^{-1} denotes inverse morphisms and $H^{-1} \circ H$ mappings of the form a morphism followed by an inverse morphism. Finally, we denote by $1T$ (resp. $2T$) the families of mappings defined by one-way (resp. two-way) finite transducers and we put in front of these abbreviations D, kV, kA, FV and FA to denote deterministic, k-valued, k-ambiguous, finite-valued or finite-ambiguous restrictions, respectively.

We denote by Reg and CF the families of regular and context-free languages (over Σ). Further by $HDTOL$ we mean the family of HDTOL languages, cf. [RS], defined as follows. Let w be a word over an alphabet Γ and $h_1, \ldots, h_k$, for some $k \geq 1$, morphisms from Γ^* into itself and f another morphism from Γ^* into Σ^*. We define the language

$$L = \bigcup_{i=0}^{\infty} f(L_i) ,$$

where

$$L_0 = \{w\}$$

$$L_{i+1} = h_1(L_i) \cup \ldots \cup h_k(L_i) \quad \text{for } i \geq 0 ,$$

and call languages L thus obtained <u>HDTOL languages</u>. It is straightforward to see that Reg $\subseteq HDTOL$, and can be shown that the families CF and $HDTOL$ are incomparable, cf. [NRSS].

Now, we formutale our problems. As earlier let Σ and Δ be two fixed finite alphabets. Further let L be a family of languages over Σ and Θ a family of partial many-valued mappings from Σ^* into Δ^*. We say that mappings σ and τ <u>universally</u>

(resp. existentially or with multiplicities) agree or are equivalent on a word $x \in \Sigma^*$ if

$$\sigma(x) = \tau(x) \quad \text{as ordinary sets} \tag{1}$$

$$(\text{resp. } \sigma(x) \cap \tau(x) \neq \phi \quad \text{whenever} \quad \sigma(x) \cup \tau(x) \neq \phi) \tag{2}$$

$$(\text{resp. } \sigma(x) = \tau(x) \quad \text{as multisets}) \tag{3}$$

Further we say that σ and τ agree (universally, existentially or with multiplicities) on a language $L \subseteq \Sigma^*$ if they do so on each of its words. We denote by

$$EP_{\forall}(\theta, L)$$

$$(\text{resp. } EP_{\exists}(\theta, L))$$

$$(\text{resp. } EP_{M}(\theta, L))$$

the problem of deciding whether two given mappings from θ are universally (resp. existentially or with multiplicities) equivalent on a given language from L. We refer these problems to as universal, existential and multiplicity θ-equivalence problems for L .

It follows immediately that for single-valued partial mappings the above three types of equivalences coincide. Observe also that in the definition of the equivalence with multiplicities the mappings σ and τ actually must be considered as mappings into the set of formal power series over nonnegative integers (augmented with ∞), i.e., into $\mathbb{N}^{(\infty)}\langle\langle\Delta^*\rangle\rangle$ in terms of [SS]. However, we shall be dealing with this notion only in connection with mappings defined by finite transducers, and since in this case the notion of "equality as multisets" meaning that the transducers must produce, for each input word, each output word equally many times is so intuitive and clear, we prefer not to go into a more formalized presentation.

3. The equivalence problem for transducers

In this section we consider the equivalence problem for different types of finite transducers. Here the equivalence means

the universal equivalence so that in our earlier formulation the problem is $EP_{\forall}(\theta,\Sigma^*)$, where θ is (the family of mappings defined by) the corresponding family of transducers. Since the domains of all the transducers defined in the previous section are regular it follows that $EP_{\forall}(\theta,\Sigma^*)$ is equivalent to $EP_{\forall}(\theta,Reg)$ for all these families of transducers.

Our aim here is to point out a borderline between decidable and undecidable equivalence problems for finite transducers. To start with we first recall that the problem for all one-way finite transducers is undecidable as shown in [FR] and at the same time even in a slightly stronger form in [Gr]:

Theorem 1. The equivalence problem for λ-free nondeterministic gsm's (sequential transducers) is undecidable.

A striking generalization of this result was proved by Ibarra in [I1]:

Theorem 2. The equivalence problem for λ-free nondeterministic gsm's (sequential transducers) with unary output alphabet is undecidable.

As regards deterministic transducers it seems to us that the decidability of the equivalence problem for these has been known for a long time, a special case is covered already in [Mo], cf. also [Bi] and [JL], but the original proof can be found nowhere. In other words, the result seems to be considered as folklore. We present here a proof which is (after knowing some elementary automata theory) very simple and which also allows some generalizations.

Theorem 3. The equivalence problem for deterministic finite transducers is decidable.

Proof. Let $M_i = \langle\Sigma,\Delta,Q_i,q_i,F_i,E_i\rangle$, for $i = 1,2$, be two deterministic finite transducers. We define an infinite state automaton $A_\infty = \langle Q,q,F,\sigma\rangle$ as follows:

$$Q = Q_1 \times Q_2 \times \Delta^{(*)}$$

$$q = (q_1,q_2,\lambda)$$

$$F = \{(q,q',\lambda) \mid q \in F_1, q' \in F_2\}$$

$$\sigma((p_1,p_2,o),a) = (q_1,q_2,u_1^{-1}ou_2) \quad \text{if}$$

$$(p_i,a,u_i,q_i) \in E_i \quad \text{for} \quad i = 1,2 \ .$$

Here $\Delta^{(*)}$ denotes the free group generated by Δ. For each q in Q let us denote by length(q) the length of the (reduced) word in the third component of q. Now, for each $k \geq 1$, let A_k be the subautomaton of A_∞ obtained from it by removing all the states q (and corresponding transitions) for which length(q) $> k$.

Clearly, A_∞ is deterministic and

$$L(A_\infty) \subseteq \mathrm{dom}(T_1) \cap \mathrm{dom}(T_2) \tag{4}$$

It also follows from the construction that T_1 and T_2 are equivalent if and only if

$$L(A_\infty) = \mathrm{dom}(T_1) = \mathrm{dom}(T_2) \tag{5}$$

But now $\mathrm{dom}(T_1)$ and $\mathrm{dom}(T_2)$ are regular and hence (5) holds (remember (4)) if and only if

$$\exists k \geq 0 \quad \text{such that} \quad L(A_k) = \mathrm{dom}(T_1) = \mathrm{dom}(T_2) \tag{6}$$

This last equivalence is a consequence of the following three facts:

(i) The minimal automaton for deterministic infinite state automaton is finite if and only if it accepts a regular language, cf. [E];

(ii) In each single step of a computation of A_∞ the length of a state can not decrease by more than a fixed constant amount depending only on T_1 and T_2;

(iii) The length of the final states of A_∞ equals 0.

From (6) we obtain a semialgorithm for the equivalence of T_1 and T_2. Since a semialgorithm for the nonequivalence is trivial our proof is complete. □

It follows immediately from the above proof that the theorem holds also in the case when the output structure is a finitely generated free group instead of such a monoid. Similar

results, even in stronger forms, have been proved in [Li1] and [Li2]. If the determinism is defined like in deterministic pushdown automata, i.e., in a state it is allowed to read the empty word provided that in this state no symbol can be read, then we obtain a wider class of transducers. For this class the equivalence problem is shown to be decidable in [BH2].

Theorem 3 was generalized for single-valued transducers in [S] and independently in [BH1]:

Theorem 4. The equivalence problem for single-valued transducers is decidable.

Next step in generalizing Theorem 3 was made in [GI] where the following result was proved:

Theorem 5. The equivalence problem for finite-ambiguous transducers is decidable.

Still one step in generalizing the above decidability results for one-way transducers was achieved recently as a consequence of a more general result of [CK3]:

Theorem 6. $EP_V(FV1T, HDTOL)$ is decidable.

Outline of the proof. The proof is based on the following two important results: (i) The validity of the Ehrenfeucht Conjecture, which states that each system of equations over a finitely generated free monoid and with a finite number of unknowns is equivalent to its finite sybsystem, cf. [K] and [AL1]. (ii) The decidability result of Makanin, which states that it is decidable whether a given equation over a finitely generated free semigroup has a solution, cf. [Mak].

In addition to these results we use techniques, cf. [CK2] or [CK3], which allow to state the fact that two considered transducers are equivalent on a given word in terms of solutions of certain systems of equations over a free monoid. In this way we associate languages with systems of equations. Further the languages we are considering, HDTOL languages, are in a certain sense morphically defined, and hence it turns out that the systems of equations associated with these languages are so simple that equivalent finite subsystems (guaranteed by the

Ehrenfeucht Conjecture) can be effectively found.

The construction of the above systems of equations depends on the k-valuedness of the transducers. So we have to be able to find such a k. This can be done by a result in [GI] (assuming that, as is the case, the transducers are finite-valued). □

Corollary 1. The equivalence problem for finite-valued transducers is decidable.

Now, it is interesting to compare Corollary 1 with Theorems 1 and 2. The characteristic feature of the transducers in Corollary 1 is that there exists an upper bound for the number of different outputs produced for single input words. Freely speaking this can be stated that the global degree of nondeterminism (with respect to outputs) is bounded. In this case the equivalence problem is decidable. On the other hand, if this degree of nondeterminism is unbounded, then the problem becomes undecidable even in quite restrictive cases as shown by Theorem 2.

On the previous lines we generalized Theorem 3 by allowing some nondeterminism. Another direction to generalize it is to consider more powerful deterministic transducers, for example two-way transducers. It was for a long time an open problem to decide whether two deterministic two-way transducers are equivalent, until it was solved by Gurari in [G1], cf. also [G2]:

Theorem 7. The equivalence problem for deterministic two-way transducers is decidable.

Recently this result was generalized in [CK2] as follows:

Theorem 8. The equivalence problem for single-valued two-way transducers is decidable.

The proof of Theorem 8 resembles that of Theorem 6, and in fact the same ideas can also be used to generalize it for k-valued or finite-valued two-way transducers, cf. [CK3].

4. Universal equivalence of mappings on languages

In this section we consider mappings which are compositions

of morphisms and inverse morphisms and study the universal equivalence of such mappings on languages, mainly on regular languages. The problem of asking whether two morphisms are equivalent on a given language was first explicitly studied in [CS], although the same problem had occurred implicitly already earlier in connection with some other problems. Indeed, the well-known DOL sequence equivalence problem, cf. [CF] or [CK1], can be stated in this form as follows: given a word w in Σ^* and two morphisms h and g from Σ^* into itself decide whether h and g are equivalent on the language $\{h^n(w) \mid n \geq 0\}$.

Concerning morphisms the following result was proved in [CS], cf. also [ACK] and [I2] where the result has been generalized:

Theorem 9. $EP_\forall(H, CF)$ is decidable.

As an evidence of the nontriviality of our Theorem 6 we note that it contains as a special case the following result which, in turn, is a proper generalization of the above mentioned DOL sequence equivalence problem.

Theorem 10. $EP_\forall(H, HDTOL)$ is decidable.

Next we turn to consider more general mappings. Based on the fact that finite one-way transducers can be simulated by morphisms and inverse morphisms, cf. e.g. [KL], the following result was deduced from Theorem 1 in [KK]:

Theorem 11. $EP_\forall(H \circ H^{-1}, \mathrm{Reg})$ is undecidable.

Actually, in Theorem 11 the family Reg can be replaced by the family $F = \{F^* \mid F \text{ is finite}\}$. Consequently, we also have:

Theorem 12. $EP_\forall(H \circ H^{-1} \circ H, \Sigma^*)$ is undecidable.

On the other hand, it was also shown in [KK] that if we reverse the order of morphisms and inverse morphisms, then the problem becomes decidable (ever for the family CF as shown in [Mao3]).

Theorem 13. $EP_\forall(H^{-1} \circ H, \mathrm{Reg})$ is decidable.

Theorems 11 and 13 are interesting in the sense that they reveal the borderline between the decidability and the undecidability in a certain setting. More precisely, let us call a partial mapping morphic if it is a composition of morphisms and inverse morphisms. Now, Theorems 11 and 13 determines sharply for which types of compositions their equivalence on regular languages can be decided.

A natural way to generalize the notion of a morphism is to consider substitutions. In this case the most important decidability question is however unanswered:

Open problem 1. Is the $EP_\forall(S,Reg)$ decidable or not?

We feel that this problem is interesting, but also difficult. There exists some support for this evaluation. First of all it can be shown, cf. [AL2], that, not only for regular languages, but for all languages the following result holds: Given a natural number $k \geq 0$ and a language $L \subseteq \Sigma^*$, then there exists a finite subset F of L such that, for any two finite substitutions σ and τ satisfying $\max \{card(\sigma(a)), card(\tau(a)) \mid a \in \Sigma\} \leq k$, σ and τ are equivalent on L if and only if they are equivalent on F. So the question is whether such an F can be found effectively for each regular language. It would be surprising if this is not the case. On the other hand, the above result does not hold even noneffectively for the regular language ab*a with respect to all finite substitutions as shown in [La].

Our conjecture is that $EP_\forall(S,Reg)$ is decidable. However, as the second evidence of its nontriviality we mention the following related result of [Mao2]:

Theorem 14. Given a regular language $L \subseteq \Sigma^*$ and two finite substitutions σ and τ on Σ^* it is undecidable whether the relation $\sigma(x) \subseteq \tau(x)$ holds for all x in L.

We conclude this section by stating a generalization of Theorem 11 due to [Mao1]:

Theorem 15. $EP_\forall(S^{-1},Reg)$ is undecidable.

5. Other types of equivalences and open problems

First we consider the existential equivalence. Intuitively it is clear that it is more difficult to decide the existential rather than the universal equivalence of two mappings. That this is really the case is seen from the following result proved in [KM], cf. Theorem 13:

Theorem 16. $EP_{\exists}(H^{-1},\mathrm{Reg})$ is undecidable.

Actually, Theorem 16 remains valid if, like in Theorem 11, Reg is replaced by F. Since the problem $EP_{\exists}(H,\mathrm{Reg})$ is trivially decidable, as a special case of that of Theorem 13, we have found also in the case of the existantial equivalence of morphic mappings on regular languages a sharp borderline between the decidability and the undecidability.

Concerning the equivalence with multiplicities we have only the following simple observation (due to T. Harju): For any finite transducer T if we add to it, for each state q, the loops (q,λ,λ,q) we obtain a transducer which produces every output with the multiplicity ∞. Hence, the problem $EP_M(1\mathcal{T},\Sigma^*)$ is undecidable by Theorem 1. However, this undecidability is based on the identity $a + \infty = \infty$ and is thus in a sense artificial. So let us consider only such transducers which do not have any transitions in $Q\times\{\lambda\}\times\{\lambda\}\times Q$. Let us denote the family of such one-way transducers by $1\mathcal{T}_e$. Obviously each transducer T in this class satisfies: the multiplicity of $y \in |T|(x)$, for any $x \in \Sigma^*$ and $y \in \Delta^*$, is bounded. On the other hand, it is also obvious that, for any one-way transducer T' satisfying this condition, there exists a transducer T" in $1\mathcal{T}_e$ such that T' and T" are equivalent with multiplicities.

Now, we state

Open problem 2. Is the problem $EP_M(1\mathcal{T}_e,\Sigma^*)$ decidable?

We again conjecture that this is the case, and further that the problem is difficult. Indeed, a solution to this problem would immediately give a solution to the equivalence problem for deterministic two-tape acceptors, which is known to be

decidable, but not easy, cf. [Bi].

We conclude this paper by discussing more our first open problem $EP_{\forall}(S,Reg)$ introduced in Section 4. We give an equivalent formulation of this problem in terms of equivalence problems for transducers. In order to be able to do this we call a one-way transducer T input deterministic if the underlying automaton of T is deterministic. (So each input deterministic transducer is a gsm.) Let us denote this family of transducers by $\mathcal{D}_i 1\mathcal{T}$. We have

Open problem 1'. Is the problem $EP_{\forall}(\mathcal{D}_i 1\mathcal{T},\Sigma^*)$ decidable?

Based on the well-known fact that the set of accepting computations of a finite automaton forms a regular set it is not difficult to prove, cf. [CK3]:

Theorem 17. The problem $EP_{\forall}(S,Reg)$ is decidable if and only if $EP_{\forall}(\mathcal{D}_i 1\mathcal{T},\Sigma^*)$ is decidable.

We feel that Theorem 17 makes our problem 1 even more interesting.

Acknowledgement. The author is grateful to T. Harju and E. Kinber for useful discussions.

References

[ACK] Albert, J., Culik II, K. and Karhumäki, J., Test sets for context-free languages and algebraic systems of equations, Inform. Control 52 (1982) 172-186.

[AL1] Albert, M. and Lawrence, J., A proof of Ehrenfeucht's conjecture, Theoret. Comput. Sci. (to appear).

[AL2] Albert, M. and Lawrence, J., Test sets for finite substitutions, manuscript (1985).

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

[Bi] Bird, M., The equivalence problem for deterministic two-tape automata, J. Comput. System Sci. 7 (1973) 218-236.

[BH1] Blattner, M. and Head, T., Single-valued a-transducers, J. Comput. System Sci. 15 (1977) 310-327.

[BH2] Blattner, M. and Head, T., The decidability of equivalence for deterministic finite transducers, J. Comput.

System Sci. 19 (1979) 45-49.

[CF] Culik II, K. and Fris, I., The decidability of the equivalence problem for DOL-systems, Inform. Control 35 (1977) 20-39.

[CK1] Culik II, K. and Karhumäki, J., A new proof for the DOL sequence equivalence problem and its implications, in A. Salomaa and G. Rozenberg (eds): The Book of L (Springer, Berlin, 1986).

[CK2] Culik II, K. and Karhumäki, J., The equivalence problem for single-valued two-way transducers (on NPDTOL languages) is decidable, SIAM J. of Comput. (to appear).

[CK3] Culik II, K. and Karhumäki, J., The equivalence of finite valued transducers (on HDTOL languages) is decidable, Proceedings of MFCS 86 (to appear).

[CS] Culik II, K. and Salomaa, A., On the decidability of morphic equivalence for languages, J. Comput. System Sci. 17 (1978) 163-175.

[E] Eilenberg, S., Automata, Languages and Machines, vol. A (Academic Press, New York, 1974).

[EY] Ehrich, R. and Yau, S., Two-way sequential transductions and stack automata, Inform. Control 18 (1971) 404-446.

[FR] Fischer, P. and Rosenberg, A., Multitape one-way nonwriting automata, J. Comput. System Sci. 2 (1968) 88-101.

[Gr] Griffiths, T., The unsolvability of the equivalence problem for ε-free nondeterministic generalized machines, J. Assoc. Comput. Mach. 15 (1968) 409-413.

[Gu1] Gurari, E., The equivalence problem for deterministic two-way transducers is decidable, SIAM J. Comput. 11 (1982) 448-452.

[Gu2] Gurari, E., Two-way counter machines and finite-state transducers, J. Comput. Math. 17 (1985) 229-236.

[GI] Gurari, E. and Ibarra, O., A note on finite-valued and finitely ambiguous transducers, Math. Systems Theory 16 (1983) 61-66.

[H] Harrison, M., Introduction to Formal Languages (Addison-Wesley, Reading, MA, 1978).

[I1] Ibarra, O., The unsolvability of the equivalence problem for ε-free NGSM's with unary input (output) alphabet and applications, SIAM J. Comput. 4 (1978) 524-532.

[I2] Ibarra, O., 2DST mappings on languages and related problems, Theoret. Comput. Sci. 19 (1982) 219-227.

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

[K] Karhumäki, J., The Ehrenfeucht Conjecture: A compactness claim for finitely generated free monoids, Theoret.

Comput. Sci. 29 (1984) 285-308.

[KK] Karhumäki, J. and Kleijn, H.C.M., On the equivalence problem of compositions of morphisms and inverse morphisms, RAIRO Inform. Théor. 19 (1985) 203-211.

[KL] Karhumäki, J. and Linna, M., A note on morphic characterization of languages, Discrete Appl. Math. 5 (1983) 243-246.

[KM] Karhumäki, J. and Maon, Y., A simple undecidable problem: Existential agreement of inverses of two morphisms on a regular language, J. Comput. System Sci. (to appear).

[La] Lawrence, J., The non-existence of finite test sets for set-equivalence of finite substitutions, EATCS Bull. 28 (1986) 34-37.

[Li1] Lisovik, L.P., Finite coverings of regular events by strong sets, Doklady of Ukrainian Academy of Sciences (1979) N 5.

[Li2] Lisovik, L.P., On solvable problems for Converters with Finite Rotary Counters, Kibernetika (1985) N 3 1-8.

[Mak] Makanin, G.S., The problem of solvability of equations in a free semigroup, Mat. Sb. 103 (1977) 147-236.

[Mao1] Maon, Y., On the equivalence of some transductions involving letter to letter morphisms on regular languages, manuscript (1985).

[Mao2] Maon, Y., Decision problems concerning equivalence of transductions on languages, Ph.D. Thesis, Tel Aviv University (1985).

[Mao3] Maon, Y., On the equivalence problem of composition of morphisms and inverse morphisms on context-free languages, manuscript (1984).

[Mo] Moore, E.F., Gedanken-experiments on sequential machines, in: Automata Studies (Princeton University Press, 1956).

[NRSS] Nielsen, M., Rozenberg, G., Salomaa, A. and Skyum, S., Nonterminals, homomorphisms and codings in different variations of OL systems. I and II, Acta Inform. 3 (1974) 357-364 and 4 (1974) 87-106.

[RS] Rozenberg, G. and Salomaa, A., The Mathematical Theory of L Systems (Academic Press, New York 1980).

[S] Schützenberger, M.P., Sur les relations rationelles in: Lecture Notes in Computer Science 33 (Springer, Berlin 1975) 209-213.

[SS] Salomaa, A. and Soittola, M., Automata-Theoretic Aspects of Formal Power Series (Springer, Berlin 1978).

Proc. IMYCS '86 October 13-17, 1986
Smolenice Castle, ČSSR

BASIC IDEAS OF SELECTIVE SUBSTITUTION GRAMMARS

H.C.M. Kleijn

Department of Computer Science University of Leiden
2300 RA Leiden The Netherlands

INTRODUCTION

In this paper a general framework for the study of rewriting systems is discussed.
After some preliminaries the concept of a selective substitution grammar is presented and motivated in Section 2. In Section 3 we introduce s-grammars as instances of selective substitution grammars. This gives rise to a simple framework still general enough to characterize in a uniform way different features of rewriting systems.
In the remainder of the paper we review the lines of research pursued until now. In Section 4 through 7 general approaches within the study of s-grammars are sketched. In Section 8 concrete classes of grammars are investigated in the framework of s-grammars, whereas in Section 9 a particular class of s-grammars, suited for an investigation of very basic properties of rewriting, is considered. Generalizations to two-dimensional and infinitary languages are briefly mentioned in Section 10. Finally, in Section 11, an extension to a general framework for the study of grammars and automata is discussed.

1. PRELIMINARIES

We assume the reader to be familiar with the basic concepts of formal language theory as, e.g., in the scope of Salomaa [27] and Rozenberg and Salomaa [25]. In addition the following notations and terminology are used.

Throughout the paper we assume that an infinite alphabet of symbols is available: all symbols that will be used are elements of the infinite alphabet $A \cup \bar{A}$, where $\bar{A} = \{\bar{a} : a \in A\}$ and A and $\bar{A}$ are disjoint. A bar appearing above a symbol indicates that the original symbol is activated. Symbols without a bar are non-activated. A consists of non-activated symbols only. In the sequel all alphabets different from $A, \bar{A}$ or $A \cup \bar{A}$ are tacitly assumed to be finite.

For a word w, $|w|$ is its length and we denote the empty word by Λ.

Let V and W be alphabets; $V,W \subseteq A \cup \bar{A}$.

(1). A total mapping h from V^* into non-empty subsets of W^* is a substitution from V^* into W^*) if $h(\Lambda) = \{\Lambda\}$ and, for all $a \in V$ and $v \in V^*$ $h(av) = h(a)h(v)$. For $K \subseteq V^*$, $h(K) = \cup\{h(v):v \in K\}$; h is a finite subsitution if h(a) is finite for all $a \in V$.

(2). Let h be a finite substitution from V^* into W^*.

(2.1). h is a finite-letter-substitution if $h(a) \subseteq W$, for all $a \in V$.

(2.2). h is a homomorphism if h(a) consists of one element for all $a \in V$; h is a coding if additionally $h(a) \subseteq W$ for all $a \in V$.

(2.3). h is a weak identity if it is a homomorphism,and for all $a \in V$, $h(a) = \{a\}$ or $h(a) = \{\Lambda\}$.

The families of all substitutions, finite substitutions, finite-letter-substitutions, homomorphisms and codings from V^* into W^* are denoted by SUB(V,W), FSUB(V,W), FLSUB(V,W), HOM(V,W), and COD(V,W), respectively.

Let $h \in SUB(V,W)$. h is non-erasing if, for all $a \in V$, $\Lambda \notin h(a)$. h is disjoint if, for all $v,w \in V^*$, such that $v \neq w$, $h(v) \cap h(w) = \emptyset$. If, for all $a \in V \cap A$, $h(a) \subseteq (W \cap A)^*$, and, for all $\bar{a} \in V \cap \bar{A}$, $h(\bar{a}) \subseteq (W \cap \bar{A})$, then h is called bar-preserving.

In the sequel we use a fixed coding iden $\in COD(A \cup \bar{A},A)$ to "remove bars". It is defined by $iden(a) = iden(\bar{a}) = \{a\}$, for all $a \in A$. The restriction of iden to subsets of $A \cup \bar{A}$ will also be denoted by iden.

A context-free grammar is specified as a 4-tuple (V,h,S,T), where V is its total alphabet, $T \subseteq V$ is its terminal alphabet, $S \in V-T$ its axiom (start-symbol) and $h \in FSUB(V-T,V)$ defines its set of productions in the following way: if $w \in h(a)$, for some $w \in V^*$ and $a \in V-T$, then (a,w) is a production and we write $(a,w) \in h$. The class of context-free grammars is denoted by CF.

An EOS system (see, e.g., [19]) is, roughly speaking, a context-free grammar in which the rewriting of terminals is allowed. We specify it as a 4-tuple (V,h,S,T), where V and T are as for context-free grammars, $S \in V$, and $h \in FSUB(V,V)$ defines its set of productions. The class of EOS systems is denoted by EOS.

A context-free grammar (or EOS system) (V,h,S,T) is propagating if h is non-erasing. The class of propagating context-free grammars (EOS systems) is denoted by Λ - CF (EPOS, respectively).

For a context-free grammar or EOS system G,the direct derivation relation $\Rightarrow_G$

is defined in the usual way. The derivation relation $\overset{*}{\underset{G}{\Rightarrow}}$ is its reflexive and transitive closure. The language of G, denoted by L(G), is defined by $L(G) = \{w \in T^* : S \overset{*}{\underset{G}{\Rightarrow}} w\}$.

The families of languages generated by context-free grammars and by EOS systems are denoted by $L(CF)$ and $L(EOS)$, respectively. Clearly $L(CF) = L(EOS)$.

A context-free grammar (V,h,S,T) is right-linear if, for all $a \in V-T$, and for all $w \in h(a)$, $w \in T(V-T) \cup T \cup \{\Lambda\}$.

The class of (propagating) right-linear grammars is denoted by (Λ-)RLIN. The family of languages generated by right-linear grammars (i.e. the family of regular languages) is denoted by $L(Reg)$.

The family of context-sensitive languages and the family of recursively enumerable languages are denoted by $L(CS)$ and $L(RE)$, respectively. We use ALL to denote the family of all languages.

2. SELECTIVE SUBSTITUTION GRAMMARS

Within formal language theory the notion of a rewriting system (or grammar) forms one of the most important tools in the study of formal languages. During the development of the grammatically oriented formal language theory numerous instances of rewriting systems have been defined, see e.g., Salomaa [27],and Dassow and Păun [3].

In 1977 Rozenberg proposed in [23] a unifying framework for rewriting systems. His aim was not to capture all existing rewriting systems in one general definition but rather to single out basic features of many kinds of rewriting systems and to define a general notion of a rewriting system based on these abstractions.

These basic features are the following:

Rewriting rules or productions that describe the replacement (substitution) of single (occurrences of) letters.

A rewriting mechanism that prescribes the use of productions in a word (selection) thus defining direct derivation steps.

A control on the composition of sequences of direct derivation steps.

A language defining mechanism.

Their abstractions when put together yield the notion of a selective substitution grammar.

We will abstain from giving the full formal definition of a selective substitution grammar, but rather describe its components on an informal basis and relate them to the basic characteristics of rewriting systems as described

above.

A selective substitution grammar is specified as a 7-tuple $G = (V,E,U,C,B,T,\psi)$.

$V \subseteq A$ is the alphabet of G.

In G the rewriting of a single symbol is formalized using the notion of a based substitution. For $A \subseteq V$, an A-based substitution is a substitution $\delta \in SUB(V \cup \bar{A},V)$ such that $\delta(\bar{a}) \subseteq V^*$, for all $a \in A$ and $\delta(a) = \{a\}$, for all $a \in V$.

In a direct derivation step in G from a word $x \in V^*$, selected occurrences in x are replaced according to some based substitution in the following way. Given an A-based substitution $\delta \in SUB(V \cup \bar{A},V)$ and a selector (language) $K \subseteq (V \cup \bar{A})^*$, where $A \subseteq V$, one considers all words $y \in K$ that are equal to x upto bars, i.e. iden(y) = x. Each such y allows a rewriting in x of all and only those occurrences in x that have been activated (occur barred) in y. The activated occurrences are replaced according to δ. Hence a word $x \in V^*$ directly derives a word $u \in V^*$, if and only if u is a result from an application of a selective substitution δ_K to x, where δ and K are as above. This means that $u \in \delta_K(x)$, where $\delta_K(x) = \cup\{\delta(y) : y \in K \text{ and } iden(y) = x\}$.

In general G has several selective substitutions which are specified in U, the set of substitution blocks of G. $U = \{\varphi_e : e \in E\}$ where E is the set of labels of G and each φ_e is a selective substitution with an underlying A_e-based substitution δ^e and a selector $K_e \subseteq (V \cup \bar{A}_e)^*$, for some $A_e \subseteq V$.

Now sequences of direct derivation steps, each using a selective substitution φ_e as described above, are composed according to the control set C of G, where $C \subseteq E^*$. For $c = e_1 \ldots e_n$, where $n \geq 1$ and $e_i \in E$, for $1 \leq i \leq n$, φ_c denotes the composition $\varphi_{e_n} \circ \ldots \circ \varphi_{e_1}$; $\varphi_\Lambda(x) = \{x\}$, for all $x \in V^*$. For $x,u \in V^*$, x derives u according to some $c \in E^*$, denoted by $x \overset{c}{\Rightarrow} u$, if $u \in \varphi_c(x)$. In G only derivations $x \overset{c}{\Rightarrow} u$ such that $c \in C$ are allowed.

The language of G, L(G) is obtained by considering all words that can be derived from words in $B \subseteq V^*$, the set of axioms (startwords) of G, and then applying to them a (partial) mapping ψ (the filter of G) from V^* into T^*, where T is the terminal alphabet of G.

Hence $L(G) = \{w \in T^* : \text{there exist a word } x \in B \text{ and a word } u \in V^* \text{ such that } x \overset{c}{\Rightarrow} u, \text{ for some } c \in C, \text{ and } \psi(u) = w\}$.

In [23] Rozenberg demonstrates the flexibility of the framework of selective subsititution grammars by showing how a variety of classes of grammars fits into it. Two of these examples are provided here.

Example 2.1. Let $G = (V,E,U,C,B,T,\psi)$ be a selective substitution grammar, where

(1). $B = \{S\}$, with $S \in V-T$, $E = \{e\}$, $C = E^*$, $T \subseteq V$, ψ is a partial identity mapping defined on T only, $U = \{\varphi_e\}$, with $\varphi_e = \delta_K$, $K = V^*\overline{V-T}V^*$ and $\delta \in FSUB(V \cup \overline{V-T},V)$.

Then G is interpreted as a context-free grammar.

(2). $B = \{w\}$, with $w \in V^*$, $C = E^*$, $T \subseteq V$, ψ is a partial identity mapping defined on T only, each φ_e in U, for $e \in E$, is of the form δ_K^e with $K = \bar{V}^*$ and $\delta^e \in FSUB(V \cup \bar{V},V)$.

Then G is interpreted as an ETOL system (see Rozenberg and Salomaa [25]). Note that if U contains only one substitution block, then we are dealing with an EOL system. □

3. s-GRAMMARS

As the notion of a selective substitution grammar provides a framework for a general theory of rewriting systems it is rather involved. In order to investigate various properties of rewriting systems it is useful to restrict this framework to a more concrete and simpler one, in which the features one is interested in are high-lighted.

The most striking aspect of selective substitution grammars is the explicit way (by selectors) of selecting the occurrences to be rewritten which is more implicit in most classes of grammars. This motivated Rozenberg and Wood in [26] to use context-free grammars with selection as special instances of selective substitution grammars in order to gain more understanding of the role of selection. In later research also the possibility of rewriting terminals is taken into account.

This has led to the introduction of s-grammars which provide a simple framework still general enough for a unified approach to the study of rewriting systems. (See Kleijn [17].) Here we introduce the notion of an s-grammar as a restricted version of a selective substitution grammar with one startletter, one substitution block, the standard filter, and implicit control.

Let $G = (V,E,U,C,B,T,\psi)$ be a selective substitution grammar and let $A \subseteq V$ be a fixed set of symbols, the active symbols of G. Furthermore, let $B = \{S\}$, with $S \in A$, $E = \{e\}$, $C = E^*$, $T \subseteq V$, ψ is a partial identity mapping defined on T only, $U = \{\varphi_e\}$ with $\varphi_e = \delta_K$, for some $K \subseteq (V \cup \bar{A})^*$ and $\delta \in FSUB(V \cup \bar{A},V)$.

Now G can be specified in the form (V,h,S,T,K), where $h \in FSUB(A,V)$ is defined by $h(a) = \delta(\bar{a})$, for all $a \in A$.

(V,h,S,T) is called the base of G and denoted by base(G), and K is called the selector of G and denoted by sel(G). We use A(G) to denote the set of active symbols of G.

Note that, for $A(G) = V-T$, base(G) is the specification of a context-free grammar and, for $A(G) = V$, it is the specification of an EOS system. In the former case we refer to G as a CF-based s-grammar and in the latter case we refer to it as an EOS-based s-grammar. If G is an X-based s-grammar, with $X \in \{CF,EOS\}$, we may also refer to it as an s-grammar.

The rewriting process in an s-grammar $G = (V,h,S,T,K)$ can easily be described: $x \in V^*$ can be rewritten if and only if K contains a word y that is equal to x upto bars. The rewriting of x is now performed by applying productions from h to exactly those occurrences in x that correspond to barred (activated) occurrences in the chosen selector word y. All other occurrences remain untouched. The language of the grammar is the set of all words over the terminal alphabet T of G that can be derived from its axiom S by iterating this procedure. Formally we have the following definitions.

Let $G = (V,h,S,T,K)$ be an s-grammar.

For $x,u \in V^*$, x directly derives u (in G) if there exists a word $y \in K$, such that iden(y) = x, and if $y = a_1 \ldots a_n$, for $a_i \in V \cup \overline{A(G)}$, $1 \leq i \leq n$, then $u = w_1 \ldots w_n$ where, for $1 \leq i \leq n$, $w_i = a_i$ if $a_i \in V$ and $w_i \in h(\text{iden}(a_i))$ if $a_i \in \overline{A(G)}$. Let $\underset{G}{\Rightarrow}$ denote the direct derivation relation in G; then $\overset{*}{\underset{G}{\Rightarrow}}$ is its reflexive and transitive closure.

The language of G, denoted as L(G), is now defined by $L(G) = \{w \in T^* : S \overset{*}{\underset{G}{\Rightarrow}} w\}$.

Example 3.1. Let $G = (V,h,S,T,K)$ be an s-grammar.

(1). $K = V^*\overline{A(G)}V^*$.

Then L(G) = L(base(G)), since the rewriting procedure described by K corresponds to the rewriting in the underlying CF-grammar (A(G) = V-T) or EOS-system (A(G) = V).

(2). $K = \bigcup_{i=1}^{n} \bar{V}_i^*$, with $V_i \subseteq A(G)$, for $1 \leq i \leq n$.

Then G corresponds to an ETOL system with (partial) tables $h_1,\ldots,h_n$ where, for $1 \leq i \leq n$, $h_i(a) = h(a)$, for $a \in V_i$, and $h_i(a)$ is not defined otherwise. □

The following important observations are immediate consequences of the definitions.

Theorem 3.1. Let G be an s-grammar.

(1). $L(G) \subseteq L(\text{base}(G))$.

(2). If $V^*\overline{A(G)}V^* \subseteq \text{sel}(G)$, where V is the alphabet of G, then $L(G) = L(\text{base}(G))$. □

A selector is a language over (a finite subset of) $A \cup \bar{A}$. Any family of languages over $A \cup \bar{A}$ will be called a family of selectors. One obtains different classes of s-grammars by varying the families of selectors and by varying the classes of bases. Let X be a class of context-free grammars or a class of EOS systems and let K be a family of selectors. Then $(X,K) = \{G : G$ is an s-grammar with base$(G) \in X$ and sel$(G) \in K\}$ and $L(X,K) = \{L(G) : G \in (X,K)\}$.

In the theory of s-grammars research is focussed on the interrelationships between classes of selectors, classes of bases and families of languages generated by s-grammars with certain bases and selectors. Before we describe the ideas underlying the research until now and some of the results obtained so far, we first discuss the relationship between CF-based s-grammars and EOS-based s-grammars in order to facilitate our later considerations.

As we have seen $L(CF) = L(EOS)$ and for many questions in formal language theory it does not matter whether or not it is allowed to rewrite terminals. In a general theory of rewriting systems, however, this difference may become important. In the case of s-grammars it is obvious that every CF-based s-grammar can be viewed as an EOS-based s-grammar which has only, say, identity productions for its terminal symbols.

Theorem 3.2. For every selector K, $L(CF,\{K\}) \subseteq L(EOS,\{K\})$. □

On the other hand in general an EOS-based s-grammar cannot directly be interpreted as a CF-based s-grammar without changing its selector. Since in a CF-based s-grammar terminal symbols cannot be activated (occur barred in the selector), also the selector of an EOS-based s-grammar has to be transformed in some way in order to arrive at a CF-based s-grammar. If a certain family K of selectors is closed under such a transformation then $L(CF,K) = L(EOS,K)$. We discuss briefly a transformation from EOS-based s-grammars to CF-based s-grammars preserving equivalence and the family of selectors involved, if that is closed under a certain operation.

Let $G = (V,h,S,T,K)$ be an EOS-based s-grammar. Let $T' = \{a' : a \in T\}$ and let $f \in FLSUB(V \cup \overline{V}, V \cup T' \cup (\overline{V-T}) \cup \overline{T'})$ defined by $f(a) = \{a\}$, for $a \in (V-T) \cup \overline{(V-T)}$, $f(a) = \{a,a'\}$, for $a \in T$, and $f(\bar{a}) = \{\bar{a}'\}$, for $a \in T$. Define the finite substitution $g \in FSUB((V-T) \cup T', V \cup T')$ by $g(a) = f(h(a))$, for $a \in V-T$, and $g(a') = f(h(a))$, for $a \in T$. Let $Z = S$, if $S \in V-T$, and let $Z = S'$, if $S \in T$. Then $H = (V \cup T', g, Z, T, f(K))$ is a CF-based s-grammar and $L(H) = L(G)$.

The transformation f applied to K is a disjoint and bar-preserving finite-letter-substitution. If K is a family of selectors that is closed under disjoint and bar-preserving finite-letter-substitutions, then we say that K is dbpfls. Hence we have the following result.

Theorem 3.3. If K is a dbpfls family of selectors then $L(CF,K) = L(EOS,K)$. □

4. FAMILIES OF SELECTORS

A natural first step in the investigation of s-grammars is to investigate how "big" a family of selectors should be in order to define a "reasonable" family of languages. (This topic is addressed in Rozenberg and Wood [26].) It turns out that when no restrictions are imposed on the selectors any language can be generated by an s-grammar.

Example 4.1. Let $T \subseteq A$ and let $L \subseteq T^*$ be an arbitrary language. The CF-based s-grammar $G = (V,h,S,T,K)$ is defined by $V = \{S\} \cup \{a' : a \in T\} \cup T$; $h(S) = \{aS : a \in T\} \cup \{a' : a \in T\} \cup (\{\Lambda\} \cap L)$ and $h(a') = \{a\}$, for all $a \in T$; $K = V^*\bar{S} \cup \{a_1 \ldots a_{n-1}\bar{a}'_n : n \geq 1, a_i \in T, \text{ for } 1 \leq i \leq n, a_1 \ldots a_{n-1}a_n \in L\}$. Then $L(G) = L$. □

Hence we have

Theorem 4.1. $L(RLIN,ALL) = ALL$. □

Corollary 4.1. $L(CF,ALL) = L(EOS,ALL) = ALL$. □

Example 4.1. also implies that for $K \in \{L(RE), L(CS), L(CF), L(Reg)\}$, $L(RLIN,K) \supseteq K$. Hence much of the generative capacity of s-grammars stems from the possibility of encoding the desired language in the selector. It is, however, desirable to define languages using objects that are less complicated than these languages themselves. In case of $L(RE)$ it turns out to be sufficient to consider only regular selectors.

Theorem 4.1. $L(CF,L(Reg)) = L(CF,L(CF)) = L(CF,L(CS)) = L(CF,L(RE)) = L(RE)$. □

Since $L(Reg)$, $L(CF)$, $L(CS)$, and $L(RE)$ are dbpfls a similar result holds for EOS-based s-grammars. When we impose additional restrictions on the bases the situation changes:

Theorem 4.3. $L(\Lambda\text{-RLIN},L(RE)) = L(\Lambda\text{-CF},L(RE)) = L(RE)$.
$L(\Lambda\text{-CF},L(Reg)) = L(\Lambda\text{-CF},L(CF)) = L(\Lambda\text{-CF},L(CS)) = L(CS)$.
$L(\Lambda\text{-RLIN},L(Reg)) \subsetneq L(\Lambda\text{-RLIN},L(CF)) \subsetneq L(\Lambda\text{-RLIN},L(CS)) = L(CS)$. □

For more detailed considerations we refer to Rozenberg and Wood [26].

5. STRUCTURAL RESTRICTIONS ON SELECTORS

As we have seen even with rather restricted families of selectors, s-grammars can still generate complicated families of languages. But some

restrictions have more influence on the language generating power than others. From Example 3.1 it follows that with selectors of the form $V^*\bar{V}V^*$ all and only context-free languages are generated whereas s-grammars with selectors of the form $\bar{V}^*$ generate the EOL languages. Such considerations lead to the question what features of selectors are responsible for the language generating power of s-grammars.

Intuitively the language generating power of a selector stems from the possibilities it has to use information from the context in the rewriting process and the possibility of blocking a derivation (by not providing a matching selector word for the current sentential form) if something goes wrong. In Rozenberg and Wood [26] some aspects of the above features are formalized and then investigated for their effects on the language generating power. In Kleijn and Rozenberg [19] this study is continued in more detail. Using context-free grammars as an example of grammars where context-information does not influence the rewriting process, and where no essential derivation-blocking possibilities are present, various "context-free" restrictions are imposed on the selectors of s-grammars. All combinations of these restrictions have been investigated. Some combinations of restrictions yield characterizations of the context-free languages, for some combinations lower- and upperbounds on the language generating power of the resulting s-grammars can be given, whereas there are also combinations which do not restrict the language generating power. Roughly four types of restrictions are distinguished in [19] and formalized as conditions to be satisfied by s-grammars.

Bar-freeness, which forbids to program the choice of particular places in a string to be rewritten.

Interspersion, which forbids to test on the immediate neighbourhood of letters.

Symbol-freeness, which forbids to distinguish between symbols that should or should not appear at particular places in a word.

Universality, which requires that every word containing an active symbol can be rewritten.

Here we only give formal definitions for symbol-freeness (which is also investigated in Section 9) and universality.

Let $G = (V,h,S,T,K)$ be an s-grammar.

G is symbol-free if, for every $w_1,w_2 \in (V \cup \overline{A(G)})^*$ and for every $a \in A(G)$ and $b \in V$, whenever $w_1\bar{a}w_2 \in K$, then $w_1\overline{A(G)}w_2 \subseteq K$, and whenever $w_1bw_2 \in K$, then $w_1Vw_2 \subseteq K$.

G is universal if, for every $w \in V^*A(G)V^*$, there exists a word $v \in K$, such that $v \neq w$ and iden$(v) = w$.

Note that the s-grammar $G = (V,h,S,T,V^*\overline{A(G)}V^*)$ is both symbol-free and universal. Hence the class of context-free languages constitutes a lowerbound on the generative power of the symbol-free and universal s-grammars. The s-grammar $G = (V,h,S,T,\overline{A(G)}^*)$ is symbol-free. If G is CF-based it is not universal. In case G is EOS-based, however, it is universal. Hence all EOL languages can be generated by symbol-free and universal EOS-based s-grammars. The next result shows that these s-grammars can even generate languages with arbitrarily complicated length sets.

Theorem 5.1. Let $R \subseteq \mathbb{N}$.

There exists a symbol-free and universal EOS-based s-grammar G such that $R = \{|w| : w \in L(G)\}$. □

For the case of CF-based s-grammars one can only prove the following theorem.

Theorem 5.2. Let $R \subseteq \mathbb{N}$

There exists a symbol-free CF-based s-grammar G such that $R = \{|w| : w \in L(G)\}$. □

This difference is explained by the following result.

Theorem 5.3. (1). All languages can be generated by universal EOS-based s-grammars. □

(2). A language is context-free if and only if it can be generated by a universal CF-based s-grammar. □

Hence in case the rewriting of terminals is not allowed, universality provides a characterization of the context-free languages. In Kleijn and Rozenberg [19] the difference between CF-based s-grammars and EOS-based s-grammars under all combinations of restrictions is further investigated. Here we stress that transformations between the two types of s-grammars as described in Section 3 of this paper are not guaranteed to preserve the restrictions imposed on s-grammars, e.g., a universal CF-based s-grammar cannot directly be interpreted as a universal EOS-based s-grammar.

As regards the bases we can add the following remarks. In Kleijn and Rozenberg [19] only propagating bases (i.e. from Λ-CF and EPOS) are considered. This, however, does not affect the results as we have stated them here. In Gonczarowski et al. [14] for some combinations of restrictions it is shown that they do not affect the language generating power even in the case that the bases satisfy additional requirements. (See also Section 7.)

6. PROPERTIES OF GENERATED LANGUAGES

Until now we have concentrated on the influence of the properties of the selector of an s-grammar on the language generated by the s-grammar. Another approach is to consider certain properties of (families of) languages and to try to find conditions on (families of) selectors guaranteeing those desired properties for the (families of) languages generated by the corresponding s-grammars. In Gonczarowski et al. [12,13] and in Chapter 5 of Kleijn [17] this line of research is pursued for closure properties. In [12,13] a wide range of language theoretical operations is considered. Then, for each of those operations, conditions on selector families are formulated which guarantee that the families of languages generated by the corresponding s-grammars are closed under this operation. A number of these general results is applied to prove that some specific families of languages are closed under certain operations. This demonstrates once more the usefulness of having a general theory of rewriting systems. Chapter 5 of [17] is based on the research presented in [12,13] and focusses on AFL closure properties.

A family of languages is called an abstract family of languages (an AFL for short), if it contains a non-empty language and is closed under each of the following operations: union, Kleene cross, non-erasing homomorphism, inverse homomorphism, and intersection with regular languages. An AFL is full if it is closed under arbitrary homomorphism.
For each of the above properties a set of conditions on families of selectors is presented guaranteeing that the families of languages generated by the corresponding classes of s-grammars have this property. The combination of conditions yields the following result.

Theorem 6.1. Let K be a dbpfls family of selectors that satisfies all of the following conditons.
(1). There exists a $K \in K$, such that $K \cap \bar{A} \neq \emptyset$.
(2). K is closed under union.
(3). K is closed under union with languages of the form $\bar{W}^*$ with $W \subseteq A$.
(4). K is closed under concatenation with languages of the form W^* with $W \subseteq A$.
Then $L(EOS,K) = L(CF,K)$ is a full AFL. □

7. BASES

Having investigated the relationship between families of selectors and families of languages generated by s-grammars using these selectors, we now turn to considerations explicitly involving the bases of s-grammars. Some

attention to the role of the bases has already been given in previous sections. In particular s-grammars with bases from CF and s-grammars with bases from EOS have been compared and possibilities of performing transformations without affecting the families of selectors or the language generating power have been considered. In fact the topic of grammatical transformations (to a certain "normal form") is a traditional one in formal language theory. In Gonczarowski et al. [14] and in Chapter 5 of Kleijn [17] which is based on [14], "standard" grammatical transformations are considered in the frame-work of s-grammars (as suggested in [26]). Whether or not a transformation can be performed within a given class of s-grammars depends on the family of selectors involved. This leads to the formulation of conditions on selector families. The results obtained are applied to specific classes of s-grammars. Here we present a result from [17].

An s-grammar (V,h,S,T,K) is binary (has a binary base), if, for all $a \in A(G)$ and $w \in V^*$, $w \in h(a)$ implies $|w| \leq 2$.
To perform a more or less standard transformation to binary bases within a certain class of s-grammars, the following notion is introduced.
Let $V \subseteq A' \cup \bar{A}$ and let $t \in A$ be such that $\{t,\bar{t}\} \cap V = \emptyset$. A substitution $g \in SUB(V, V \cup \{t,\bar{t}\})$ such that, for $a \in V \cap A$, $g(a) = t^* a t^*$ and, for $\bar{a} \in V \cap \bar{A}$, $g(\bar{a}) = \bar{t}^* \bar{a} \bar{t}^*$ is called an r-substitution (for V and t).

Theorem 7.1. Let K be a dbpfls family of selectors that is closed under union and r-substitution. Then, for every $G \in (EOS,K)$ there exists an equivalent binary $G' \in (EOS,K)$.

At this point one should notice the difference between the existence of a normal form for bases and the possibility of changing bases without leaving a certain family of selectors. Note that by Example 4.1, for every language L, a binary s-grammar exists that generates L. In Gonczarowski et al. [14] it is shown that certain standard restrictions as, e.g., chain-freeness, imposed on the bases (even when combined with additional restrictions on the selectors as discussed in Section 5) do not affect the language generating power of the whole class of s-grammars. From Example 4.1 (slightly modified) an even stronger normal form for bases follows: one fixed base suffices to generate all languages (over a fixed terminal alphabet). This leads to the following notions, which are introduced and discussed in Rozenberg and Wood [26].

Let Y be a class of context-free grammars or EOS systems and let K be a family of selectors. Let $T \subseteq A$.
$G \in Y$ is said to be K-universal for Y modulo T if

$L(\{G\},K)=\{L \subseteq T^* : L \in L(Y,K)\}$.

$K \in K$ is said to be Y-universal for K modulo T if

$L(Y,\{K\}) = \{L \subseteq T^* : L \in L(Y,K)\}$.

(The interested reader may also look up Gonczarowski et al. [14] for the related notion of s-generator.)

Now one can investigate what conditions on a selector family K guarantee the existence of a K-universal base. From Example 4.1. it follows that for every alphabet $T \subseteq A$ there exists a context-free grammar (EOS system) that is K-universal for CF (EOS) modulo T, where K is any family closed under union with regular languages and "renaming" . This can be applied to specific families of selectors as, e.g., ALL,L(RE),L(CS),L(CF), and L(Reg).The next result (from Rozenberg and Wood [26]) is proved using transformations to a fixed base .

Theorem 7.2. Let K be a family of selectors, that is closed under union and finite substitutions. Let $T \subseteq A$. There exists a context-free grammar that is K-universal for CF modulo T. □

For the grammar-universality of families of selectors we do not present results as it is a more restricted notion than the selector-universality of bases. A fixed selector prohibits the possibility of encoding directly the languages to be generated and moreover it establishes an upperbound on the number of non-terminals that can be used actively in the base.

8. SPECIFIC SELECTORS

In this section we present an example of the study of concrete "rewriting modes" prescribed by specific families of selectors. Such research is presented in Ehrenfeucht et al.[11], Kleijn and Rozenberg [20] and continued in Kleijn and Rozenberg [21],Ehrenfeucht et al.[10] and Subramanian [30]. Sequential and parallel rewriting modes are investigated and compared in the framework of s-grammars, together with a new "in-between" continuous way of rewriting. Using context-free grammars (selectors of the form $V^*\overline{V\text{-}TV}^*$) and EOL systems (selectors of the form $\bar{V}^*$) as extreme examples of sequential and parallel rewriting three classes of s-grammars are introduced. Sequential grammars (rewriting only one occurrence in a derivation step), parallel grammars (rewriting all occurrences in a derivation step), and continuous grammars (rewriting a continuous segment in a derivation step).

Let $n \geq 1$.

The family of n-sequential selectors, denoted by nS, is defined by

$nS = \{\bigcup_{i=1}^{n} X_i^*\bar{Y}_iZ_i^* : X_i,Y_i,Z_i \subseteq A, \text{ for } 1 \leq i \leq n\}$.

The family of n-parallel selectors, denoted by nL, is defined by
$nL = \{\bigcup_{i=1}^{n} \bar{Y}_i^* : Y_i \subseteq A, \text{ for } 1 \leq i \leq n\}$.
The family of n-continuous selectors, denoted by nC, is defined by
$nC = \{\bigcup_{i=1}^{n} X_i^+\bar{Y}_i^+Z_i^* : X_i,Y_i,Z_i \subseteq A, \text{ for } 1 \leq i \leq n\}$.
An s-grammar G is called sequential (parallel, continuous) if
sel(G) $\in$ nS(nL,nC), for some $n \geq 1$.
Note that nS,nC, and nL are dbpfls families of selectors. Hence, by Theorem 3.3, $L(EOS,K) = L(CF,K)$, for $K \in \{nS,nC,nL\}$.
Much emphasis has been given to the investigation of the language generating power of these classes of s-grammars both in relation to one another and in relation to known classes (see [10],[11],[20]). Not all questions have been solved yet. In [12,13] and [17] an application of the results on closure properties (see Section 6) yields that the family of languages generated by continuous grammars is an AFL. This has also independently and directly been proved in [30]. In [20] and [21] also the role of erasing productions is considered and combinations of sequential, continuous and parallel selectors are investigated. The sequential, continuous and parallel modes of rewriting are investigated further (in [20]) by subjecting them to certain fundamental restrictions as context-symmetry and selection determinism. This brings to light essential differences between sequential, continuous and parallel grammars and yields new characterizations for several known classes of languages.

9. PATTERN GRAMMARS

Within the framework of s-grammars it is possible to consider special classes of s-grammars which in their turn are sufficiently "broad" to allow a unified approach to the basics of rewriting processes. The class of pattern grammars (introduced in Kleijn and Rozenberg [19]) forms such a concrete framework. A pattern grammar is actually a symbol-free s-grammar (see Section 5). In such an s-grammar the symbols occurring in the selector are not relevant. The only thing that matters is whether or not they occur activated (barred). Hence the selector controls the rewriting only by prescribing "rewriting patterns" which can be viewed as consisting of two symbols: 1 for "rewrite" and 0 for "do not rewrite". Varying the rewriting patterns leads to very different rewriting systems. For instance, a context-free grammar uses rewriting patterns from 0^*10^* (i.e. rewrite one occurrence) and an EOL system uses patterns from 1^* (i.e. rewrite all occurrences).

In the remainder of this section 0 and 1 are distinguished symbols.

A pattern grammar is a construct $G = (V,h,S,T,K)$ where $\underline{base}(G)=(V,h,S,T)$ is a context-free grammar or an EOS system and $\underline{sel}(G) = K \subseteq \{0,1\}^*$.
Let $s_{V,A(G)} \in FSUB(\{0,1\}, V \cup \bar{A}(G))$ be defined by $s_{V,A(G)}(0) = V$ and $s_{V,A(G)}(1) = \overline{A(G)}$. Then $s(G) = (V,h,S,T,s_{V,A(G)}(K))$ is the symbol-free s-grammar corresponding to G. The direct derivation relation and derivation relation in G are inherited from s(G) and $L(G) = L(s(G))$.
Note that a symbol-free s-grammar $H = (V,h,S,T,K)$ corresponds to the pattern grammar $s^{-1}(H) = (V,h,S,T,s^{-1}_{V,A(H)}(H))$ and $ss^{-1}(H) = H$. Hence symbol-free s-grammars and pattern grammars specify the same objects. However, since the selectors of pattern grammars do not involve the names of symbols they facilitate a general approach: One language $K \subseteq \{0,1\}^*$ determines a family of selectors $\{s_{V,A}(K) : A \subseteq V \subseteq A\}$. Any language over $\{0,1\}$ will be called a pattern selector. Using observations similar to those in Section 3, it can easily be seen that, for pattern grammars - even in the case of one fixed selector - the difference between EOS bases and CF bases can be discarded. For a family K of pattern selectors, $L(pK) = \{L(G):G$ is a pattern grammar with $\underline{sel}(G) \in K\}$ Let Pat denote the family of all pattern selectors and let RegPat denote the family of all regular pattern selectors. From Theorem 5.2 it follows that L(pPat) contains arbitrarily complicated languages. In Kleijn and Rozenberg [19] and [22] the generative power of regular pattern grammars (pattern grammars with a regular selector) is investigated. This leads to the following results.

Theorem 9.1. (1). L(pRegPat) $\subseteq L$(RE).
(2). For every $L \in L$(RE), $L\,c^5 \in L$(RegPat), where c is a new symbol.
(3). For every $L \in L$(RE), there exists a weak identity g and a propagating regular pattern grammar G such that $L = g(L(G))$. □

Hence even the simple class of regular pattern grammars generates "almost" the recursively enumerable languages. This generative power seems to stem from the "counting" ability of regular pattern selectors. In order to destroy this ability two additional restrictions are considered in [22].
$K \subseteq \{0,1\}^*$ is commutative if, for all $x,y \in \{0,1\}^*$, $x01y \in K$ if and only if $x10y \in K$.
$K \subseteq \{0,1\}^*$ is prefix closed if, for all $x,y \in \{0,1\}^*$, $xy \in K$ implies that $x \in K$.
The family of commutative and prefix closed regular patterns is denoted by CPRegPat.

Theorem 9.2. L(EOL) $\subsetneq L$(pCPRegPat) $\subsetneq L$(CS). □

Interesting examples of rewriting patterns are $0^*1^k0^*$ and $0^*(10^*)^k$, $k \geq 1$, which determine "context-free" grammars in which in every derivation step k (adjacent or scattered) symbols are rewritten in parallel. (The rewriting of the axiom is "free".) It is easy to see that, for $k \geq 2$, the patterns $0^*(10^*)^k$ give rise to non context-free languages, as, e.g. $\{a_1^n \ldots a_k^n : n \geq 1\}$. For the adjacent case it remained for some time an open problem whether or not $L(p\{0^*110^*\})$ contains non context-free languages (see [18]). This problem has recently been solved by Dahlhaus and Gaifman [2], who showed that $L(p\{0^*110^*\})$ contains non EOL languages.

Theorem 9.3. $L(CF) \subsetneq L(p\{0^*110^*\})$. □

In Gonczarowski and Shamir [15] and Gonczarowski and Warmuth [16] parsing algorithms are developed and the complexities of the membership problems are investigated for families $L(p\{0^*1^k0^*\})$ and $L(p\{0^*(10^*)^k\})$, $k \geq 1$.

10. GENERALIZATIONS

The flexibility of the framework of selective substitution grammars is once more demonstrated in the work of Siromoney and Subramanian [29] and of Siromoney and Dare [28]. In [29] selective substitution array grammars are introduced which provide a unifying framework for many of the two dimensional array grammars in the literature. In [28] a method is presented to generate infinite words using selective substitution grammars. This method is compared with some well-known ways of defining languages of infinite words. Relations between several infinitary language families obtained from selective substitution grammars are established and closure properties and decidability questions are studied.

11. GRAMMARS AND AUTOMATA

In formal language theory one can distinguish next to the grammatical approach an automata based approach to the study of formal languages. As with grammars numerous instances of automata have been defined in the literature. By singling out basic features of automata one may construct a framework for a general theory of automata. However, such a close look at automata may also lead to the insight that grammars and automata are very closely related. Such considerations have motivated the introduction of a unifying framework for grammars and automata. In Rozenberg [24] coordinated table selective substitution systems (cts systems, for short) were introduced as

a unifying framework for both grammars and automata. This framework is an extension of the framework of s-grammars and is based on the rewriting of vectors of words rather than single words. Here we discuss a simplified version of cts systems. For the full framework and an extensive number of examples the reader is referred to [24].

A cts system is a construct $H = (G_1,\ldots,G_n;R)$, $n \geq 1$, where, for $1 \leq i \leq n$, $G_i = (V_i,h_i,S_i,T_i,K_i)$ is an s-grammar, and R is a set of rewrites each of which is of the form $U = (U_1,\ldots,U_n)$ with $U_i \subseteq \{(a,w): w \in h_i(a) \text{ and } a \in V_i\}$. Hence, for each $1 \leq i \leq n$, U_i is a subset of the set of productions defined by h_i.

Given $x = (x_1,\ldots,x_n)$ and $y = (y_1,\ldots,y_n)$ where $x_i,y_i \in V_i^*$, for $1 \leq i \leq n$, we say that x directly derives (or computes) y in H (using the rewrite U), denoted by $x \underset{H}{\Rightarrow} y$, if for every $1 \leq i \leq n$, x_i directly derives y_i in G_i using productions from U_i only. The reflexive and transitive closure of $\underset{H}{\Rightarrow}$ is denoted by $\overset{*}{\underset{H}{\Rightarrow}}$. The language L(H) of H is defined by

$L(H) = \{w \in T_1^* : (S_1,\ldots,S_n) \overset{*}{\underset{H}{\Rightarrow}} (w,\Lambda,\ldots,\Lambda)\}$.

In [24] various ways of defining languages of cts systems are discussed. One can interpretate one dimensional cts systems as (s-) grammars and more dimensional systems as automata with the first component as input and the other components as auxiliary (e.g., storage) devices. This motivates the above definition: a computation has ended only if the storage is empty.

Until now especially the following types of s-grammars have been used in the framework of cts systems.

Let $G = (V,h,S,T,K)$ be an s-grammar

G is RL, if base(G) $\in$ RLIN and $K = T^*\overline{(V-T)}$;

G is RB, if base(G) $\in$ EOS and $K = V^*\bar{V}$;

G is OL, if base(G) $\in$ EOS and $K = \bar{V}^*$;

G is OS, if base(G) $\in$ EOS and $K = V^*\bar{V}V^*$;

G is OS^2, if base(G) $\in$ EOS and $K = V^*\bar{V}\bar{V}V^*$.

The class of cts systems $H = (G_1,\ldots,G_n;R)$ with G_i of type X_i, where $X_i \in \{RL,RB,OL,OS,OS^2\}$, for $1 \leq i \leq n$, is denoted by $(X_1,\ldots,X_n)$. The families of languages generated by cts systems from $(X_1,\ldots,X_n)$ are denoted by $L_n(X_1,\ldots,X_n)$.

As in the framework of s-grammars one can investigate various families of selectors and their influence on the families of languages generated by cts sytems using these selectors. In cts systems selectors can be used in a "direct mode" (on the first coordinate) and in an "indirect mode" (on another

coordinate). In [24] it has been shown that the relative power of a family of selectors depends on the mode in which the selectors are used.

Theorem 11.1. $L_1(RB) = L(Reg)$, $L_1(OS) = L(EOS) = L(CF)$, and $L_1(OL) = L(EOL)$. □

This implies that $L_1(RB) \subsetneq L_1(OS) \subsetneq L_1(OL)$. If we use the same families of selectors at the second coordinate, the situation changes as can be seen from the next theorem. In all three cases we assume that the first component is RL which corresponds to the standard use of an input tape. $L(PN)$ in the statement of the theorem denotes the family of languages defined by labelled marked Petri nets with final zero marking (see Aalbersberg and Rozenberg [1]).

Theorem 11.2. $L_2(RL,RB) = L(CF)$, $L_2(RL,OS) = L(PN)$, and $L_2(RL,OL) = L(Reg)$. □

This implies that $L_2(RL,OL) \subsetneq L_2(RL,RB)$, $L_2(RL,OL) \subsetneq L_2(RL,OS)$ and $L_2(RL,RB)$ and $L_2(RL,OS)$ are incomparable. The equality $L_2(RL,OS) = L(PN)$ has been proved in Aalbersberg and Rozenberg [1]. In that paper the relationship between (classes of) Petri nets and (classes of) cts systems is investigated. In addition cts systems from (RL,OS^2) are investigated.

Theorem 11.3. $L_2(RL,OS^2) = L(RE)$. □

It is interesting to compare this result with the remarks in Section 9 on $L_1(OS^2) = L(p\{0^*110^*\})$.

The main part of the research in the framework of cts systems until now is devoted to (RL,RB) systems. (see Ehrenfeucht et al. [4] through [9]). (RL,RB) systems, usually called coordinated pair systems or cp systems, model the classical push-down automata (pda's for short). The notion of a cp system is simpler than that of a pda and the framework of cp systems gives rise to new results on the behaviour of pda's. Also new proofs for already known results can be provided without reference to other constructs like context-free grammars.

In [4] a normal form for cp systems is established yielding the so-called real-time cp systems. In the proof of this result rather than the grammatical Greibach normal form the structure of computations in cp systems is considered.

Much emphasis is given to the study of computations in cp systems. An important tool is the Exchange Theorem (see [7]) that describes how to swap subcomputations between computations in a cp system. In [8] and [9] this tool is used to investigate the possibilites of obtaining pumping properties of context-free languages via the analysis of computations in cp systems.

This leads in particular to an analysis of the structure of Dyck words. The correspondence between the structure of Dyck words and computations in cp systems can then be used to derive pumping lemma's. In [6] a survey of results is given.
In [5] the use of the "memory" (the RB component) of a cp system is investigated yielding as an overall conclusion that the evaluation of the memory behaviour depends strongly on the observation method chosen.

ACKNOWLEDGEMENT

The author is indebted to H.J. Hoogeboom for a careful reading of a first version of this paper.

REFERENCES

[1] Aalbersberg, IJ.J. and G. Rozenberg, CTS systems and Petri nets, Theoretical Computer Science 40 (1985), 149-162.

[2] Dahlhaus, E. and H. Gaifman, Concerning two-adjacent context-free languages, Theoretical Computer Science 41 (1985), 169-184.

[3] Dassow, J. and Gh. Păun, Regulated rewriting in formal language theory, in preparation.

[4] Ehrenfeucht, A., Hoogeboom, H.J., and G. Rozenberg, Real-time coordinated pair systems, Dept. of Comp. Sci., Univ. of Colorado at Boulder, Tech. Rep. CU-CS-259-83, 1983.

[5] Ehrenfeucht, A., Hoogeboom, H.J., and G. Rozenberg, On the active and full records of the use of memory in right-boundary grammars and push-down automata, to appear in Theoretical Computer Science.

[6] Ehrenfeucht, A., Hoogeboom, H.J., and G. Rozenberg, On coordinated rewriting, Lect. Notes in Comp. Sci. 199 (1985), 100-111.

[7] Ehrenfeucht, A., Hoogeboom, H.J., and G. Rozenberg, Computations in coordinated pair systems, to appear in Fundamentae Informaticae.

[8] Ehrenfeucht, A., Hoogeboom, H.J., and G. Rozenberg, Coordinated pair systems. Part I: Dyck words and classical pumping, to appear in R.A.I.R.O. Informatique Théorique.

[9] Ehrenfeucht, A., Hoogeboom, H.J., and G. Rozenberg, Coordinated pair systems. Part II: Sparse structure of Dyck words and Ogden's lemma, to appear in R.A.I.R.O. Informatique Théorique.

[10] Ehrenfeucht, A., Kleijn, H.C.M., and G. Rozenberg, Adding global forbidding context to context-free grammars, Theoretical Computer Science 37 (1985), 337-360.

[11] Ehrenfeucht, A., Maurer, H., and G. Rozenberg, Continuous grammars, Information and Control 46 (1980), 71-91.

[12] Gonczarowski, G., Kleijn, H.C.M., and G. Rozenberg, Closure properties of selective substitution grammars. Part I, International Journal of Computer Mathematics 14 (1983), 19-42.

[13] Gonczarowski, G., Kleijn, H.C.M., and G. Rozenberg, Closure properties of selective substitution grammars. Part II, International Journal of Computer Mathematics 14 (1983), 109-134.

[14] Gonczarowski, G., Kleijn, H.C.M., and G. Rozenberg, Grammatical constructions in selective substitution grammars, Acta Cybernetica 6 (1983), 239-269.

[15] Gonczarowski, J. and E. Shamir, Pattern selector grammars and several parsing algorithms in the context-free style, Journal of Computer and Systems Sciences 30 (1985), 249-273.

[16] Gonczarowski, J. and M.K. Warmuth, Applications of scheduling theory to formal language theory, Theoretical Computer Science 37 (1985), 217-243.

[17] Kleijn, H.C.M., Selective substitution grammars based on context-free productions, Ph.D.Thesis, University of Leiden, 1983.

[18] Kleijn, H.C.M. and G. Rozenberg, Problems P 111-113, EATCS Bulletin 26 (1985), 240-241.

[19] Kleijn, H.C.M. and G. Rozenberg, Context-free like restrictions on selective rewriting, Theoretical Computer Science 16 (1981), 237-269.

[20] Kleijn, H.C.M. and G. Rozenberg, Sequential continuous and parallel grammars, Information and Control 48 (1981), 221-260. Corrigendum,ibidem 52 (1982), 364.

[21] Kleijn, H.C.M. and G. Rozenberg, Multi grammars, International Journal of Computer Mathematics 12 (1983), 177-201.

[22] Kleijn, H.C.M. and G. Rozenberg,On the generative power of regular pattern grammars, Acta Informatica 20 (1983), 391-411.

[23] Rozenberg, G., Selective substitution grammars (Towards a framework for rewriting systems). Part I: Definitions and examples, Elektronische Informationsverarbeitung and Kybernetik 13 (1977), 455-463.

[24] Rozenberg, G., On coordinated selective substitutions: towards a unified theory of grammars and machines, Theoretical Computer Science 37 (1985), 31-50

[25] Rozenberg, G. and A. Salomaa, The mathematical theory of L systems, Academic Press, New York, 1980.

[26] Rozenberg, G. and D. Wood, Context-free grammars with selective rewriting, Acta Informatica 13 (1980), 257-268.

[27] Salomaa, A., Formal languages, Academic Press, New York, 1973.

[28] Siromoney, R. and V.R. Dare, On infinite words obtained by selective substitution grammars, Theoretical Computer Science 39 (1985), 281-295.

[29] Siromoney, R. and K.G. Subramanian, Selective substitution array grammars, Information Sciences 25 (1981), 73-83.

[30] Subramanian, K.G., On the language class of continuous grammars, unpublished manuscript (1983).

Proc.IMYCS '86 October 13-17,1986
Smolenice Castle,ČSSR

SOME RECENT RESTRICTIONS IN THE DERIVATION OF CONTEXT-FREE GRAMMARS

Gheorghe PĂUN

University of Bucharest
Division of Systems Studies
Str. Academiei 14, 70109 Bucureşti
ROMANIA

We discuss here three classes of regulation mechanisms for context-free grammars, all three introduced in the eighties. The first one, the valence grammars in Păun [13], associates numbers to production rules and accepts as correct only derivations with a certain total valence. The second mechanism is a variant of random context restriction (strings instead of symbols in context sets) and it has been proposed by Kelemen [9]. The third restriction is a new one and it is based on the so-called walk language associated to a grammar.

1. Introduction

The regulated rewriting is a very important (rich in notions, results, problems and applications) area of formal language theory. Proofs of this assertion can be found in the forthcoming monograph by Dassow, Păun [4], where almost all the known regulation devices are presented.

The domain is not new: the first known restriction in derivation, the matrix one, already counts more than two decades (Abraham [1]). However, new restrictions still appear. We discuss here three of the recently introduced ones (definitions, examples, no-proof results, open problems). They are the valence grammars in Păun [13], the semi-conditional grammars of Kelemen [9] (see also Păun [14]) and the new, unpublished yet, walk restricted grammars.

The idea of this last regulation mechanism is the following: take a context-free grammar and interpret the derivation process in an automata-type manner, that is consider a "rewriting head" which scans the current string and replaces certain nonterminals by right hand members of corresponding rules, then moves again and so on. The "walk" of this rewriting head can be described by a language, appropriately codifying the three basic actions it does: move to the right, move to the left, rewrite. Imposing restrictions to this language (to be given, as in a regular control grammar, for instance), we can obtain more variants of such a "walk restricted grammar". Generally, they have a great generative capacity (characterizations of context sensitive languages are obtained); some of them are strongly similar to the selective substitution grammars of Rozenberg (see Kleijn [10] for detailed references).

In what follows, the reader is assumed familiar with formal language theory basic notions and results, including rudiments of regulated rewriting (see, for instance, Salomaa [17]). Some notations: V^* is the free monoid generated by V, λ is the unity of V^*, $|x|$ is the length of x, RE, CS, CF, REG are the four families of languages in Chomsky hierarchy, LIN is the family of linear languages and MLIN is the family of metalinear ones. The components of a Chomsky grammar will be denoted $G = (V_N, V_T, S, P)$, with the nonterminals in V_N specified by capitals and the terminals in V_T by small letters.

2. Valence grammars

Definition 2.1. An additive valence grammar is a construct $G = (V_N, V_T, S, P, v)$, where $G' = (V_N, V_T, S, P)$ is a usual Chomsky grammar and $v : P \longrightarrow Z$ (Z is the set of integers). For a derivation

$$D : S \xRightarrow{r_1} w_1 \xRightarrow{r_2} w_2 \xRightarrow{r_3} \cdots \xRightarrow{r_n} w_n$$

in G', we define

$$v(D) = \sum_{i=1}^{n} v(r_i)$$

The language generated by G is

$$L(G) = \{ x \in V_T^* \mid D : S \overset{*}{\Longrightarrow} x \text{ in } G',\ v(D) = 0 \}$$

Replacing Z by Q_+ (i.e. the set of positive rational numbers), the addition by multiplication and the condition $v(D) = 0$ by $v(D) = 1$, we obtain the multiplicative valence grammars.

Example 2.1. Consider the grammars G_i, $i = 1, 2$, identified by the next rules:

$r_1 : S \longrightarrow aS,\ v_1(r_1) = 1$ and	$r_1 : S \longrightarrow aS,\ v_2(r_1) = 2$
$r_2 : S \longrightarrow aA,\ v_1(r_2) = 0$	$r_2 : S \longrightarrow aA,\ v_2(r_2) = 1$
$r_3 : A \longrightarrow bA,\ v_1(r_3) = -1$	$r_3 : A \longrightarrow bA,\ v_2(r_3) = 3$
$r_4 : A \longrightarrow b,\ v_1(r_4) = 0$	$r_4 : A \longrightarrow bB,\ v_2(r_4) = 1$
	$r_5 : B \longrightarrow cB,\ v_2(r_5) = 1/6$
	$r_6 : B \longrightarrow c,\ v_2(r_6) = 1.$

We obtain

$$L(G_1) = \{ a^n b^n \mid n \geq 1 \} \quad \text{(additive valences)}$$

$$L(G_2) = \{ a^n b^n c^n \mid n \geq 1 \} \quad \text{(multiplicative valences)}$$

Denote by AV(X) (MV(X)) the families of languages generated by additive (multiplicative, respectively) valence grammars of type X, X a class in Chomsky hierarchy. The above examples show that AV(REG) contains non-regular languages, and MV(REG) contains non-context-free languages. In what follows, REG stands for right-linear grammars and the grammars can possibly contain λ-rules.

The following results were proved in Păun [13] (some new proofs are given in Dassow, Păun [4]) and in Gheorghe [6]:

THEOREM 2.1.

(i) The families AV(REG), MV(REG) can be characterized in terms of the one-way nondeterministic finite automata with addition/multiplication and without equality in Ibarra et al. [8].

(ii) The families MV(X) equal the families of unordered generalized vector grammars of type X in Cremers, Mayer [2],

[3].

(iii) $X \subset AV(X) \subset MV(X)$, $X \in \{CF, LIN, REG\}$, strict inclusions.

(iv) $AV(REG) \subset AV(LIN) \subset AV(CF)$,
$MV(REG) \subset MV(LIN) \subset MV(CF)$,
$AV(REG) \subset CF$, strict inclusions.

(v) The families in the next pairs are incomparable:
AV(REG) and LIN, MV(REG) and CF,
MV(REG) and AV(CF), MV(REG) and LIN,
MV(REG) and MLIN, AV(LIN) and CF,
AV(LIN) and MLIN, MV(LIN) and CF.

Considering the above characterizations (and the results in Cremers, Mayer [2], [3] and Ibarra et al. [8]) as well as by ad-hoc proofs, many closure properties were obtained for valence grammars. We do not discuss them here, but we present some results of Marcus, Păun [11], concerning an extension of valence restriction to gsm mappings.

Definition 2.2. An additive valence gsm is a system $g = (K, I, O, s_0, F, P, v)$, where $g' = (K, I, O, s_0, F, P)$ is a usual gsm (with the moves in P specified as rewriting rules, $sa \longrightarrow xs'$, $s, s' \in K$, $a \in I$, $x \in O^*$) and $v : P \longrightarrow Z$. The valence $v(D)$ of some rewriting

$$D : ys_1a_1a_2\ldots a_nz \overset{r_1}{\Longrightarrow} yx_1s_2a_2\ldots a_nz \overset{r_2}{\Longrightarrow} \ldots$$

$$\ldots \overset{r_{n-1}}{\Longrightarrow} yx_1\ldots x_{n-1}s_na_nz \overset{r_n}{\Longrightarrow} yx_1\ldots x_ns_{n+1}z,$$

$z \in I^*$, $y \in O^*$, $r_i : s_ia_i \longrightarrow x_is_{i+1} \in P$, $1 \leqslant i \leqslant n$, is defined by

$$v(D) = \sum_{i=1}^{n} v(r_i)$$

and, for $w \in I^*$,

$$g(w) = \{ z \in O^* \mid \text{there is } D : s_0w \overset{*}{\Longrightarrow} zs_f,\ s_f \in F,\ v(D) = 0 \}$$

A similar definition holds for multiplicative valence gsm's: replace Z by Q_+, addition by multiplication and $v(D) = 0$ by $v(D) = 1$.

We denote by AGSM the class of additive valence gsm's,

and by MGSM the class of multiplicative valence gsm's. By AGSM(X), MGSM(X) we denote the families of languages obtained by translating a language in the family X by mappings in AGSM, MGSM, respectively. Write $AGSM^n(X)$, $MGSM^n(X)$ for n times iterated such translations.

The following results were proved in Marcus, Păun [11]:

THEOREM 2.2.

(i) $AV(X) = AGSM(X)$,

$MV(X) = MGSM(X)$, $X \in \{CF, LIN, REG\}$.

(In this way a new characterization of vector languages is obtained, as the image of context-free languages by multiplicative valence gsm mappings.)

(ii) The class AGSM is not closed under composition and the families AV(X), $X \in \{CF, REG\}$, are not closed under additive valence gsm mappings.

(iii) The class MGSM is closed under composition (therefore MV(X), $X \in \{CF, REG\}$, are closed under multiplicative valence gsm mappings).

(iv) $MGSM^n(X) = MGSM(X)$, $n \geqslant 1$, $X \in \{CF, REG\}$,

$AGSM^n(REG)$, $n \geqslant 2$, are incomparable with CF,

$AGSM^n(X) \subseteq MGSM(X)$, $n \geqslant 1$, $X \in \{CF, REG\}$.

Open problems.

Q1. Which are the relations between MLIN and AV(REG), MV(LIN) ?

Q2. The families $AGSM^n(X)$, $n \geqslant 1$, $X \in \{CF, REG\}$, define two hierarchies which lie in between AV(X) and MV(X). Are these hierarchies infinite ? We expect an affirmative answer.

Q3. Here we considered valence grammars (and gsm's) involving the particular groups $(Z, +, 0)$ and $(Q_+, \cdot, 1)$. What about considering arbitrary groups ? For instance, can we obtain an infinite hierarchy of language families taking the groups $(Z^n, +, (0, 0, \ldots, 0))$, $n \geqslant 1$? Denote by $AV_n(X)$, $n \geqslant 1$, $X \in \{CF, REG\}$, the family of languages generated by additive valence

grammars of the form $G = (V_N, V_T, S, P, v)$, $v : P \longrightarrow \longrightarrow Z^n$. We obtain (Gheorghe, Păun [7]):

(i) $AV(X) = AV_1(X)$,

(ii) $AV_n(X) \subseteq AV_{n+1}(X)$, $n \geqslant 1$,

(iii) $\bigcup_{n \geqslant 1} AV_n(X) = MV(X)$, $X \in \{CF, REG\}$,

therefore the hierarchies $AV_n(X)$ lie in between $AV(X)$ and $MV(X)$, $X \in \{CF, REG\}$, respectively. We feel that these hierarchies are infinite too.

3. Semi-conditional grammars

Kelemen [9] has proposed the following type of regulated mechanism, with AI motivation: add to each rule $A \longrightarrow x$ in a given grammar $G = (V_N, V_T, S, P)$ a string w over $V_G = V_N \cup V_T$ and apply this rule only for rewriting strings which have w as substring. Such a restriction is similar to the conditional one (Friš [5], Păun [12]), where a language is added to each rule and the rule is applied to strings in the associated language, as well as to random context grammars (Van der Walt [18]), in which each rule has a set Q of permitting symbols and a set R of forbidding symbols, the rule being applied only to strings which contain all symbols in Q and no symbol in R. A generalization of Kelemen grammars were considered in Păun [14], under the name of semi-conditional grammars.

Definition 3.1. Let i, j be two natural numbers. A semi-conditional grammar of degree (i, j) is a system $G = (V_N, V_T, S, P)$, where V_N, V_T, S are as in a usual grammar and P is a finite set of production rules of the form $(A \longrightarrow x, \alpha, \beta)$, where $A \longrightarrow x$ is a context-free rule, $\alpha = \emptyset$ or $\alpha \in V_G^*$, $|\alpha| \leqslant i$, and $\beta = \emptyset$ or $\beta \in V_G^*$, $|\beta| \leqslant j$. Such a rule can be applied to a string w if and only if α (if $\alpha \neq \emptyset$) is a substring of w and β (if $\beta \neq \emptyset$) is not a substring of w. (When $\alpha = \emptyset$, $\beta = \emptyset$, then the rule can be applied without restrictions.)

We denote by $SK(i, j)$, $i \geqslant 0$, $j \geqslant 0$, the family of languages generated by λ-free context-free semi-conditional gram-

mars of degree (i, j); when λ-rules are allowed, a super - script λ is added.

The following results were proved in Păun [14]:

THEOREM 3.1.

(i) Both families SK(1, 0), SK(0, 1) contain non-semilinear languages, hence they include strictly the family CF.

(ii) $SK(1, 1) \subset CS$, strict inclusion.

(iii) $SK(2, 1) = CS = SK(1, 2)$,

$SK^{\lambda}(2, 1) = RE = SK^{\lambda}(1, 2)$.

(iv) $SK_{left}(1, 0) = CS = SK_{left}(0, 2)$,

$SK^{\lambda}_{left}(1, 0) = RE = SK^{\lambda}_{left}(0, 2)$

(the subscript left indicates the restriction to leftmost derivations in the usual sense).

To a semi-conditional grammar one can impose a further regulating device, for instance, the order restriction of Friš [5] (introduce a partial order of rules and use the maximal applicable rules for rewriting the current string), the programmed restriction (Rozenkrantz [15]), the regular control (Salomaa [16]) or the matrix restriction (Abraham [1]). We shall add the letters O, P, C, M in the front of SK(i, j) in order to denote the corresponding families of languages, respectively. As it is expected, new characterizations of CS and RE families are obtained in this way. Please note that we do not use appearance checking features in programmed ($\varphi(r) = \emptyset$ for all rules), regular control and matrix grammars ($F = \emptyset$).

THEOREM 3.2.

$XSK(2, 0) = CS = XSK(0, 2)$, $X \in \{O, P, C, M\}$,

$XSK^{\lambda}(2, 0) = RE = XSK^{\lambda}(0, 2)$, $X \in \{O, P, C, M\}$.

A similar result can be obtained also when considering the semi-conditional restriction imposed to matrix grammars (matrices with an associated pair (α, β) as above; the whole matrix is applied only to strings containing α and not containing β, etc.).

Also for this regulation mechanism some <u>problems</u> have remained open:

Q4. Are SK(i, 0), SK(0, i), $i \geqslant 2$, strictly included into CS ? (Remember points (ii) and (iii) of Theorem 3.1.) Is $SK_{left}(0, 1)$ strictly included into CS ?

Q5. Are the inclusions $SK(1, 0) \subseteq SK(1, 1)$, $SK(0, 1) \subseteq$ $\subseteq SK(1, 1)$ proper ? Which relations there are between SK(0, i) and SK(i, 0), $i \geqslant 1$?

4. Walk restricted grammars

Consider a context-free grammar $G = (V_N, V_T, S, P)$. When in some string $w = x_1Ax_2Bx_3$ we first rewrite the A occurrence and then the B occurrence, we can say that the "writing head" of the grammar has moved from A to B. We can thus think in terms of automata even when dealing with grammars. We shall formalize this in the following way:

Definition 4.1. Let $G = (V_N, V_T, S, P)$ be a context-free grammar and consider a derivation D according to G,

$$D : S = w_0 \Longrightarrow w_1 \Longrightarrow \dots \Longrightarrow w_n \in V_T^*$$

The "grammar scanner" is initially positioned on S and for w_j, $j \geqslant 1$, it is positioned according to the next rules:

1. If $w_i \Longrightarrow w_{i+1}$, $w_i = x_1Ax_2$, $w_{i+1} = x_1yzx_2$, $x_1, y, x_2 \in$ $\in V_G^*$, $z \in V_G$, and the scanner is positioned on A in w_i, then the scanner is positioned on z in w_{i+1}.
2. If $w_i \Longrightarrow w_{i+1}$ as above, the used rule was $A \longrightarrow \lambda$ and the scanner is positioned on A in w_i, then in w_{i+1} the scanner is positioned on z in $x_2 = zx_2'$ when $x_2 \neq$ $\neq \lambda$, or on z in $x_1 = x_1'z$ when $x_2 = \lambda$, $x_1 \neq \lambda$; the scanner is "lost" when $w_{i+1} = \lambda$.

The "walk" of the grammar scanner can be codified as follows. If, according to the above definition, the scanner is positioned on z in w_i, $w_i = y_1zy_2Ax_2$, and this occurrence of A is rewritten in $w_i \Longrightarrow w_{i+1}$, then we say that the scanner has been moved k steps to the right, $k = |y_2A|$. When $w_i =$

$= x_1 A y_1 z y_2$ we say that the scanner has been moved k steps to the left, $k = |Ay_1|$.

Let us denote by 0 the action of rewriting (using a rule), by 1 the scanner moving for a step to the right and by 2 the scanner moving for a step to the left. We write

$$walk(w_i, D) = 1^k, \quad walk(w_i, D) = 2^k$$

in the above cases, respectively. Thus, the scanner walk (including the rewritings) in the derivation D will be described by the string

$$walk(D) = 0\ walk(w_1, D)\ 0\ walk(w_2, D) \ldots 0\ walk(w_{n-1}, D)\ 0$$

In this way, a language

$$walk(G) = \{ walk(D) \mid D \text{ is a derivation in } G \}$$

can be associated to the grammar G.

Example 4.1. Clearly, if G is a linear grammar, then walk(G) is a regular sublanguage of $\{0, 2\}^*$.

Take now the metalinear grammar G with rules

$$S \longrightarrow AB,\ A \longrightarrow Aa,\ B \longrightarrow bB,\ A \longrightarrow a,\ B \longrightarrow b$$

We have

$$L(G) = \{ a^n b^m \mid n \geqslant 1,\ m \geqslant 1 \}$$

$$walk(G) \cap 0(2^+01^+0)^* = \{ 020102^3 01^3 0 \ldots 2^{2k+1} 01^{2k+1} 0 \mid k \geqslant 1 \}$$

therefore the language walk(G) is not context-free.

It is easy to see that the language walk(G) is context sensitive for each context-free grammar G. As the language walk(G) is similar in some sense with the Szilard language associated to G, it is interesting to examine it as a goal *per se*. We shall not insist on this direction here, but we shall define a regulating mechanism on this basis (in the same way as the regular control grammars are defined starting from Szilard languages).

Definition 4.2. A *regular walk grammar* is a system $G = (V_N, V_T, S, P, C)$, where $G' = (V_N, V_T, S, P)$ is a usual context-free grammar and C is a regular language over $\{0, 1, 2\}$. The language L(G) is

$$L(G) = \{x \in V_T^* \mid \text{there is a derivation } D : S \overset{*}{\Longrightarrow} x \text{ in } G' \text{ such that walk}(D) \in C\}$$

We denote by RW the family of languages generated by regular walk λ-free context-free grammars; when λ-rules are allowed, we write RW^{λ}.

The inclusions

$$RW \subseteq CS, \quad RW^{\lambda} \subseteq RE$$

can be proved by a standard construction. The following examples will show that the inclusion $CF \subset RW$ is proper (moreover, RW contains non-semilinear languages).

Example 4.2. Consider the grammar G with the rules

$$S \longrightarrow AB, \; A \longrightarrow aAb, \; B \longrightarrow cB, \; A \longrightarrow ab, \; B \longrightarrow c$$

and with the regular language

$$C = 02(01^+02^+)^*$$

It is easy to see that all correct terminal derivations must be of the form

$$S \Longrightarrow AB \Longrightarrow aAbB \Longrightarrow aAbcB \Longrightarrow a^2Ab^2cB \Longrightarrow a^2Ab^2c^2B \Longrightarrow \ldots$$

$$\ldots \Longrightarrow a^nAb^nc^nB \Longrightarrow a^{n+1}b^{n+1}c^nB \Longrightarrow a^{n+1}b^{n+1}c^{n+1}$$

hence

$$L(G) = \{a^nb^nc^n \mid n \geq 1\}$$

Example 4.3. Consider the grammar G with the rules

$$S \longrightarrow BAAc, \; B \longrightarrow bB, \; B \longrightarrow b, \; A \longrightarrow AAc, \; A \longrightarrow a$$

and with the regular language

$$C = 02^3(00(1^+0)^+3^+)^*$$

Let us remark that the substrings 00 of strings in C imply the use of rules $B \longrightarrow bB$, $B \longrightarrow b$ (after using $A \longrightarrow AAc$ the scanner is positioned on c). Moreover, after using two times the rule $B \longrightarrow bB$ (thus introducing two occurrences of b), the scanner goes to the right and at least a rewriting is performed (the substring 1^+0); then we return to the left symbol B and the process is reiterated. In consequence, the strings in L(G) are of the form $b^{2n}w$ with $w \in \{a, c\}^*$, $n + 1 \leq |w|_a \leq 2^n$, $n \leq |w|_c \leq 2^n - 1$ ($|w|_z$ is the number of symbol z occurrences

in w). This language is not semilinear.

Some variants of the walk restricted grammars could be of interest. For instance, instead of 0 in the walk control language we can consider a nonterminal; the rewriting specified by 0 must now consists of rewriting the corresponding nonterminal. Another possibility is to replace 0 by a production rule label and to use this rule at the corresponding step of a derivation.

Example 4.4. Consider a context sensitive grammar G in Kuroda normal form and let r : $AB \longrightarrow CD$ be a rewriting rule in G. Construct a context-free regular walk grammar G' introducing the associated rules

$$A \longrightarrow A_r,\ A_r \longrightarrow C,\ B \longrightarrow B_r,\ B_r \longrightarrow D$$

and considering the string AA_r1BB_r (instead of 00100) as a substring of the associated walk control language. Clearly, in this way the grammar G' can simulate the rule r, hence we obtain $L(G) = L(G')$.

A similar result is true for the case when we replace the 0 occurrences by rule labels, therefore these variants of walk restricted grammars characterize the context sensitive languages (recursively enumerable languages, when λ -rules are used).

Of course, the study of regular walk grammars needs much further efforts. Here are some open problems and research topics which seem to deserve our attention:

Q6. Is the inclusion $RW \subseteq CS$ a proper one ? Compare the family RW with other families obtained by regulated rewriting.

Q7. What about considering a context-free walk language ? What about adding appearance checking features ? (Mark some occurrences of 0 in the strings of the walk language C and use them in the appearance checking manner, that is ignore them when no rewriting is possible in this place.)

References

1. S. Abraham, Some questions of phrase-structure grammars, Comp. Ling., 4 (1965), 61 - 70.

2. A.B. Cremers, O. Mayer, On matrix languages, Inform. Control, 23 (1973), 86 - 96.

3. A.B. Cremers, O. Mayer, On vector languages, Proc. Symp. Summer School Math. Found. Comp. Sci, High Tatras, 1973.

4. J. Dassow, Gh. Păun, The regulated rewriting in formal language theory, Akademie Verlag, Berlin (in press).

5. I. Friš, Grammars with partial ordering of rules, Inform. Control, 12 (1968), 415 - 425.

6. M. Gheorghe, Linear valence grammars, manuscript, 1986.

7. M. Gheorghe, Gh. Păun, Two (infinite ?) hierarchies of vector languages, Bull. of the EATCS (submitted).

8. O.H. Ibarra, S.K. Sahni, C.E. Kim, Finite automata with multiplication, Th. Comp. Sci., 2 (1976), 271 - 294.

9. J. Kelemen, Conditional grammars. Motivations, definition and some properties, Proc. Conf. Aut. Lang. Math. Syst., Salgotarjan, 1984.

10. J. Kleijn, Selective substitution grammars based on context-free productions, Doctoral Dissertation, Univ. of Leiden, 1983.

11. M. Marcus, Gh. Păun, Valence gsm mappings, Bull. Math. Soc. Sci. Math. R.S. Roumanie (in press).

12. Gh. Păun, On the generative capacity of conditional grammars, Inform. Control, 43 (1979), 178 - 186.

13. Gh. Păun, A new generative device: valence grammars, Rev. Roum. Math. Pures Appl., 25 (1980), 911 - 924.

14. Gh. Păun, A variant of random context grammars: semi-conditional grammars, Th. Comp. Sci., 41 (1985).

15. D. Rozenkrantz, Programmed grammars and classes of formal languages, Journal of the ACM, 16 (1969), 107 - 131.

16. A. Salomaa, On some families of formal languages obtained by regulated derivations, Ann. Acad. Sci. Fenn., Ser. A1, 1970, 479.

17. A. Salomaa, Formal languages, Academic Press, New York, London, 1973.

18. A.P.J. Van der Walt, Random context languages, Symp. on Formal Lang. at the MFI Oberwolfach, 1970.

Proc.IMYCS '86 October 13-17,1986
Smolenice Castle,ČSSR

KLEENE'S THEOREM REVISITED:THE POWER OF MECHANISMS

Jacques Sakarovitch

Laboratoire d'Informatique Théorique et Programmation

C.N.R.S. Paris

SUMMARY

One of the basic results in automata theory is certainly the theorem of Kleene : the one which states that the class of formal languages that are accepted by finite automata coincides with the class of languages that can be defined by regular expressions. No surprise then that this result has been given several (equivalent) statements and proved in several (slightly different) ways.

The analysis of these statements show that Kleene's theorem consists indeed in two different propositions that are better distinguished when one tries to generalize the result. The first aim of this paper is to state explicitely a possible generalization of Kleene's theorem from formal languages - that is the semi-rings of subsets of free monoids - to a class of semi-rings called here Kleene semi-ring. We call mechanism the corresponding generalization of automata. A mechanism is basically a finite graph with edges labelled by elements of a Kleene semi-ring. One then defines the result of a mechanism, which corres-

ponds to the language accepted by an automaton, and one has

Theorem 1 : Let S be a subset of a Kleene semi-ring K. The regular closure of S is equal to the class of elements of K that are result of mechanisms the entries of which belong to S.

The roots of Theorem 1 may be found in the thesis of Walljasper and in the books of Conway and Eilenberg.

But the main purpose of the paper is to show how this theorem on mechanisms, Kleene's theorem indeed, appears at other places in automata theory. We give here there examples.

The first example, a straight forward one, is a theorem by Elgot and Mezei which states that rational relation between free monoids are realized by finite state transducers. The second example is somewhat more involved. After the necessary framework has been set up, it appears that theorem 1 directly implies the Chomsky normal form for context-free grammars. In the last example, theorem 1 is used to give a characterization of regular sets in free products. This is the basis of the characterization of the free partially commutative monoids the regular sets of which form a Boolean algebra and are all unambiguous.

Proc.IMYCS '86 October 13-17,1986
Smolenice Castle,ČSSR

BASIC COMPLEXITY ANALYSIS OF HYPOTHESIS FORMATION: GUHA-Style,with Implications for A.I.Applications

Frederick N. Springsteel
University of Missouri/Columbia
Computer Science Department,Columbia Missouri 65211 USA
and
Fulbright Lecturer at University Novi Sad, SFRY

1 INTRODUCTION TO GUHA-STYLE HYPOTHESIS FORMATION

Mechanized hypothesis formation of certain general logical forms has been realised in a multilevel system, by the Czechoslovak artifical intelligence project ongoing since 1965, known as the "GUHA Project" [2,3]. The current overall concept, GUHA-80, is a long-term effort to construct an intelligent analyser of large data sets in the spirit of parallel logic processing [3]. In particular, GUHA-80 applies AI techniques to help expedite data analysis which is "exploratory" (EDA), as opposed to "confirmatory" data analysis. The latter merely confirms a hypothesis once a researcher forms one; the former searches for all patterns, of various types, in the data. MHF as done in the GUHA Circle is necessarily working on an open-ended goal:

> Given large, selected data sets as matrices over a particular domain with dozens of relevant properties as columns and hundreds of observed individuals as rows, the goal is to discover all interesting non-trivial [logical/statistical] implications and/or associations that are plausible hypotheses about the domain, using the properties as unary predicates [2].

"Interesting" is given meaning based on user options but is generally an ill-defined term, as befits an open ended task. There are copious discussions of this cognitive concept in the GUHA literature [2,3,4,5]. It suffices here to note that it is dealt with in the GUHA-80 project, but we will focus on other aspects. Each GUHA run allows the user to re-define "interesting".

Their current work develops experimental, but key, kernels of the GUHA-80 master plan which in its full

extent will be an agenda-driven, rule-based, multilevel AI-system for EDA using parallel processing, with a facet-rich, frame-like knowledge representation structure [3,6]. In many respects its symbolic reasoning is patterned on that of Lenat's AM project [7]. However, EDA deals with real-world empirical data, not pure mathematics. The GUHA-80 system will choose its parallel processes from a wide range of modular algorithms:

- DATA_REDUCTION,
- SIMILARITY_MATRIX,
- CHI_SQuare,
- BMDP3F,
- CLUSTERING (See [3] for a list of GUHA procedures.)

One significant motivation for the overall project is the AI-enhancement of huge statistical software packages, e.g. BMDP, SPSS, SAS, SURVO, etc. [8], for the benefit of ordinary users. These packages are powerful and complex and are somewhat user-oriented, but they usually cannot advise the user about what tool to select from the many possibilities, nor what parameters to use, because there are so many different user situations and variables. Users who are not statisticians, the majority, often need counsel on what software tool should be used in their particular situations.

The first parts of GUHA-80 are approaching this by developing intelligent systems which guide their users in the choice of next procedure to run and its parameters. These new systems draw heavily upon the experience of the expert systems MYCIN and SACON [9,12]. The latter advises users of a multi-part package, MARC.

Another AI/analytic system, done within SUMEX, is the RX project, which is like the EMYCIN with the addition of a statistics package [17]; see Figure 1. According to its descriptors, the RX system is admittedly primitive in that it mainly cross-tabulates each domain property versus all others, for significant statistics. It reports very few new results, possibly due to this restriction on form and length of output hypotheses.

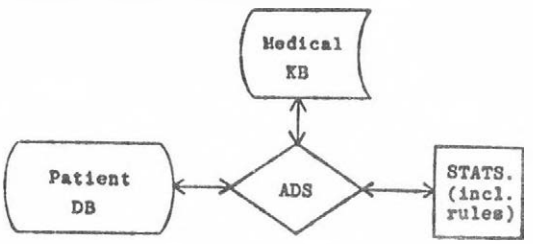


Figure 1. MYCIN/RX-type Automated Discovery System

2 EXPLORATORY DATA ANALYSIS VIA HYPOTHESIS FORMATION

Recent work on the GUHA-80 objectives has been focused on a prototype expert system, G-QUANT, and on an intelligent multi-statistic evaluator, ASSOC [5]. These kernel procedures deal with (1) the choice of a particular "quantifier", or statistic, from a variety of available modules,

and then (2) the detailed evaluation of the chosen statistical or logical formulae over the user's data matrix. There is considerable interfacing between these two modules, and opportunity for positive feedback in their repeated invocation. They form a synergistic pair, like SACON and MARC.

In order to run ASSOC, one must first set the form of quantifier to be used, e.g. an independence test for two or more properties, like Fisher's Test, or perhaps one of a class of (non-standard) implications. The expert system G-QUANT serves to help make this choice. Present implementations utilize six (6) distinct forms, which go way beyond one-by-one crosstabs. GUHA hypothesis forms have always been complex enough to relate from two to six different domain properties. Associations of compound (two-way) attributes with a third property are common.

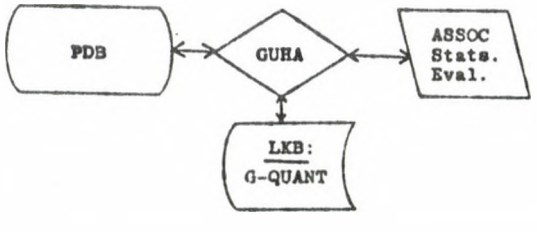


Figure 2. GUHA's Mechanized Hypothesis Former

2.1 THE GUHA procedure G-QUANT:

This GUHA module operates more like EMYCIN, without embedded domain knowledge, but with the addition of heuristic rules that give in effect a Logical/statistical Knowledge Base, the expert system G_QUANT; see Figure 2. GUHA runs typically produce much output (unless cleverly restricted), including some novel results [2], but also very many trivial, known or otherwise less interesting hypotheses. G_QUANT was inspired by EMYCIN, as a naked inference engine, but developed independently of it and RX.

The consulting system G-QUANT asks questions of the user like "What is the size of your data matrix?", and then assigns categories, like "The matrix is: Small/Medium/Large." Then this system employs the technique of backwards chaining, as in SACON and MYCIN, using its knowledge base of heuristic rules of the form:

"IF (condition-1) THEN (condition-2) WITH CERTAINTY (factor)".

Both of condition-i (i=1,2) are true/false propositions related to data analysis, and factor is a knowledge base assigned probability.

For example:

IF (the user wants the results to be decision rules) THEN (the class of the quantifier is IMPLicational) WITH CERTAINTY (1.0);

IF (the kind of quantifier is Symmetric) AND (the matrix size is Small) THEN (the identity of the quantifier is FISHER) WITH CERTAINTY (0.9);

Actually, to control complexity, G-QUANT does not use a numeric

range of "factor" from 0.0 to 1.0, but rather a discrete range of seven values of "c-degrees": from -3 (certainly not) to +3 (certainly yes); 0 means unknown. These c-degrees are assigned to the basic rules, as provided by user evidence, then propagated via the net of rules using the reverse logic of backward chaining, to recommend a quantifier of highest certainty.

2.2 Forming Hypotheses: ASSOC

While this procedure is not an expert system, it is the target of the expert system G-Quant. ASSOC is a GUHA procedure that processes binary and/or n-ary, i.e. finite-valued, nominal data in a "smart" if not intelligent way. It fosters wise choices of input parameters and uses clever default values, to guide novice users. It can also accept real-variable properties and convert them to range-value categories, so that all data is nominal.

ASSOC generates and tests hypotheses. For all six quantifiers these are binary associations of the form:

A∿∿S ("A is associated with S")

where A, S are compound propositions for which the atoms are the domain's properties. Non-atomic propositions are formed as disjunctions or conjunctions of signed atoms, and the operator ∿∿ is to be one of several (generalized) associations, called "quantifiers" [13]. Moreover, conditioned associations of the form (A∿∿S)/C may be generated as hypotheses, where C is a prior condition for A∿∿S. Only sentences with disjoint components A, S[, C] will ever be generated; "A" is called the antecedent, "S" the succedent, "C" the condition [14].

In a given run of ASSOC, the quantifier is fixed and so is the condition C (if any); A and S vary. Each generated hypothesis is tested as a statement about the data matrix, by standard methods of evaluation, including cases of missing data. When C is chosen, A∿∿S is evaluated on the submatrix which omits all rows not staisfying C. If no missing data is detected in the matrix, a proposition pair A,S determines a 4-fold contingency table:

a = # of rows satisfying A & S,
b = # of rows satisfying A & -S,
c = # of rows satisfying -A & S,
d = # of rows satisfying -A & -S.

|a b|
|c d|

The six quantifiers now used in ASSOC include three symmertric ones:

SIMPLE (association),
FISHER (independence test), and
CHI_SQ(uare statistic),

each of which satisfies a*d > b*c ("coincidence dominates difference"), as well as other defining conditions. The three non-standard, IMPLication quantifiers are:

FIMPL - Founded almost IMPLication,
LIMPL - Lower critical almost IMPLication, and
UIMPL - Upper critical almost IMPLication.

The definitions of these non-standard quantifiers are rather special: for example, (A FIMPL S) is valid iff a > BASE and a/(a+b) > CPROB, given the parameters BASE,CPROB.

EXAMPLE: Suppose the antecedent A conjoins property #2 and the negation of property #19 and S is property #20 [Suppose these domain properties are

respectively: 30-year-chain-smoker, age under 60, and lung cancer]. Then validity holds for

(P2 & -P19) FIMPL P20,

if A is true in 50 rows and both A and S are true in at least 40 of those rows, where BASE = 40 and CPROB = 0.80. In words, the hypothesis: "A 30-year-chain-smoker over 60 is likely to have lung cancer" is true for 80% of the (limited) sample.

Matrices with missing data are processed using one of 3 possible semantics: "secured" (the default), "deleting", or "optimistic"; see [13]. These use, instead, a nine-fold contingency table for three-valued logic. Other quantifiers are defined and discussed in [14]. The planned maintenance of the new systems G-QUANT and ASSOC allows for the addition of new quantifiers, as long as they are "founded" and, preferably, "associational"; see [14].

[The programming for the ASSOC system was done by P. Hajek, I. Hlaveseva, B. Louvar, D. Pokorny, A. Sochorova and E. Tschernoster. The implementation of the G_QUANT expert system was in PL/I, by Marie Hajkova, using IBM 370.]

2.3 Control of Complexity of GUHA/HF Computations

To supress output of hypotheses implied by already found ones, ASSOC runs use one or both of two logical rules: "Symmetry" for the symmetric quantifiers, and "Improvement" (either "strict" or "conservative") for all six. Improvements of $A \sim\sim S$ would only increase its validizing statistics. The need for GUHA processes to preserve strict limits on combinatorial explosion, and principles for doing so, were noted by Springsteel, and by Pudlak & Springsteel [16,15].

Later versions of ASSOC, which will allow more quantifiers, are planned to utilize some of the complexity-induced principles for bounding two-valued logic searches. These principles can best be discussed here in the context of a search for Disjunctions of at most n signed (and distinct) predicates, which disjunctions are to be true for each individual row in the data sample, where n is the total number of atomic properties. I.e., we assume all basic propositions are in their Disjunctive Normal Forms.

We shall denote an instance of this problem by $D_<(M)$, where M is the given mxn data matrix, or simply by $D_<$ for the general algorithmic problem. The first principle allows less output:

EXTENSION: If one disjunction, say d = P1|...|...|Pk, is true in M, then there are at least 2^{n-k} different disjunctions of the maximum length, n, that are true in M. [The latter are the extensions of d by each sign pattern on the (n-k) signed atoms: $(+1-)P_{k+1}$, ..., $(+1-)P_n$.]

Thus, $D_<(M)$ is true if and only if the instance $D_{max}(M)$ is true of maximum length disjunctions. The second principle is:

EXCLUSION: A non-trivial disjunction of d of maximum length n is false over matrix M if and only if the n-tuple of 0's and 1's which negates the Pi's signs is in M.

Thus, $D_{max}(M)$ is true if and only if at least one possible row is missing from M, and a true d can be found by probing for a missing row, linearly. Note that the Principles of Extension and Exclusion imply that not only is D_{max} solvable in polynomial time, over a "search space" of 2^n maximum length disjunctions, but so is $D_<$, over a much larger search space. For example,

d = P1| -P2| P3

is false only when any row <0,1,0, ..., ..., ...> is present in M; this is easily checked. But, if d is thereby found true, so are all extensions of

d to maximum length.

2.3.1 Example A: Short Disjunctions Can Play "Hard-to-Get"

Seven people are asked six Y/N questions, including sex (P3, where "1" signifies male), and five either-or preferences. Each person answers independently, but an easy analysis shows there to be several patterns of responses common to all seven. This follows from the Exclusion Principle in general, because 7 is much smaller than $2^n = 2^6 = 64$; thus, many "falsifying rows" must be missing from M. Suppose the responses are:

M:

Respondees	Preferences P1	P2	P3	P4	P5	P6
R1	0	0	0	1	1	1
R2	0	0	1	1	1	0
R3	0	1	0	1	0	1
R4	0	1	1	1	0	0
R5	1	0	0	0	1	1
R6	1	0	1	0	1	0
R7	1	1	0	0	0	1

Since 7 is actually less than 2^3, in any 3 columns we can apply Exclusion to find true disjunctions of only 3 properties. Using the four possible sequences of consecutive columns, we can find, in this special sample, exactly one length-three pattern true for every individual and each sequence of columns:

-P1|-P2|-P3, -P2|-P3|P4, -P3|P4|P5, P4|P5|P6

The fourth disjunctive pattern here says that everyone answered at least one of the last three questions "Yes". By the Principle of Extension, each of these patterns can be extended arbitrarily by signed predicates that are missing in it, to any length 4, 5, or 6. However, it is not necessary to output these "seen-at-a-glance" consequences of the algorithmically discovered length-3 patterns.

Now, one can try to use Extension to ask if any disjunctions shorter than length three could be true in the M above, meaning length two since no column is all 1's. If any length-2 disjunction were true, it would have two arbitrary extensions of length three using any extra column. By the above results, from an exhaustive search only of consecutive properties, we know that this does not happen for two adjacent columns. Indeed, M contains a full complement of 0's and 1's (four distinct rows) in any pair of ADJACENT columns! (M is minimal such.)

But how can we know if ANY two columns are full without looking at all pairs? It turns out that we can't in this case, unless we notice that M has "opposite" columns, modulo every third one, e.g., P1 = -P4, etc. Hence, P1 | P4, P2 | P5, and P3 | P6 are true over M, as are the same with all negated predicates. However, in some sense we had to look at arbitrary pairs of the n columns, or "n choose k" combinations, which becomes exponential for k > 2.

2.3.2 Example B: Minimal Seven Rows Without Length-2 Truths

Suppose the survey used above was seen to be concocted, and then refined to ask the same seven people six different 0/1 preferences. Call these changed questions C1 through C6. Answers may be as follows:

M':

Respondees	Preferences C1	C2	C3	C4	C5	C6
R1	0	0	0	0	0	0
R2	0	1	0	1	0	1
R3	1	0	1	0	1	0
R4	1	1	1	1	1	1
R5	0	1	1	0	1	1
R6	1	0	1	1	0	1
R7	1	1	0	1	1	0

A perceptive table-reader (i.e., exploratory data analyst) will realize that some pairs of respondees now fall into patterns like the questions did before: R1 = -R4, R2 = -R3. The latter two responded with opposite "alternate-the-answers" strategies. But this has little to do with finding true length-2 disjunctions, IF any exist. To find the latter, the agent will need to look at each pair of columns, to see if they always agree or always disagree. It turns out that M' has no true length-two's, and seems to be minimally so. For cases like this, it appears "length k" examples will take exponential time.

It turns out that no length-2 disjunctions are true in M', but clearly many length-3's are, since 7 < 8. E.g., C1|C2|-C3, meaning C3 implies (C1|C2). Thus the design of the new questionnaire is superior: some sample of size 7 (these folks) has no simple pattern of length less than 3 true in M'. [These examples generalize.] Therefore, systems like RX, which seek only length-two valid statements will find nothing true in this M'.

2.3.3 Example C: Truth Comes in All Sizes, Negatively

The previous examples used the fact that with only $m < 2^{n/2}$ rows there must be true disjunctions of length n/2, in ANY n/2 columns of a binary matrix. It is possible with more than $2^{n/2}$ rows that there will still be short, true disjunctions, e.g., some column may be all 1's or all 0's. Thus a matrix with 6 columns and as many as 32 (or, $n^6/2$) rows, the full 5-column binary matrix with an extra column of 0's, has true disjunctions of ALL lengths, but no purely POSITIVE-predicate disjunctions at all! But, when m increases to as many as $16 = 2^*2^{n/2}$ rows, we can find special examples, like Example B, with no true length-n/2 formulae. (See [16].)

These examples show the difficulty, and almost the complexity, of search for true disjunctions in simple two-valued matrices. Other work by this author demonstrates that even very tractable two-valued problems can instantly turn into NP-complete problems when a third logic value is introduced. Also, in general, the problems are harder when the search is for associational quantified hypotheses, but few quantifiers have been studied. Furthermore, certain HF problems seem to fall into natural intermediate complexity classes: their known time bounds are non-polynomial, non-exponential functions such as

$$N^{\log N}.$$

See [16] for a review of these difficulties; it shows the Czechs' limitations on size of desired outputs to be wisely efficient.

2.4 Selected Results in Hypothesis Formation Complexity Analysis

Here we examine certain results concerning the EXISTENCE of desired disjunctive form hypotheses true over given 2-valued (or, 3-valued) data matrices. As above, we shall denote the probelm of algorithmically determining whether such hypotheses exist for given conditions by:

Notation — Problem

D_{max} — Given any M, is there a maximum length disjunction true in M?

$D_{<}$ — Given any M, is there a disjunction of length at most 1(M) true in M?

D_{par} — Given any M and any parameter $k \leq 1(M)$, is there a disjunction of length k true in M?

$D_{1/2}$ — Given any M, is there a disjunction of length at most 1(M)/2, true in M?

These probelms can also be posed for Positive disjunctive forms, so denoted by a superscript '+'. We assume all disjunction sought are of elementary form (no predicate occurs twice), all predicates are unary properties and are considered universally quantified (over all rows in the matrices). Obviously, such types of problems include as special cases simple implications:

$(P1 \mid -P2) \mid P3 \equiv (-P1 \,\&\, P2) \Rightarrow P3.$

Therefore, even length three disjunctions are of more generality than the problems considered in the RX project.

2.4.1 Two-valued Disjunctive Existence Results

I. D^+_{max} is in P; in fact it is solvable in linear time and in log n space.

Proof: There is only one d; test it!

Corollary: $D^+_<$ is in P.

Proof: Principle of Extension makes these equivalent. [X]

II. D_{max} is in P.

Proof: Use the Principle of Exclusion to probe for any missing row in M, in linear time and logarithmic space.[X]

Corollary: $D_<$ is in P, with time and space as above.

III. $D_{1/2}$ is in P. [Modify the proof IV, below.]

Corollary: Given a fixed k, the D_{par} problem is in P.

IV. D_{par} is solvable, all k, in $N^{\log N}$ time, $(\log N)^2$ space.

Proof: Given M with m rows, and $k \leq n = 1(M)$, if: (a) $m < 2^k$ then the Exclusion principle produces true disjunctions over any k columns; so, the answer is "Yes"; (b) $m \geq 2^k$, then generate all length-k d's in some numeric order and test on M. The number of such d is at most $(n|k)*2^k < (2n)^k = 0(_n\log m)$. Say "Yes" if any.[X]

[Notice that the median case, $k = 1(M)/2$, is $D_{1/2}$.]

V. D^+par is NP-complete, as is $D^+_{1/2}$.

Proof: Reduce this one to the NODE_COVER problem; cf. [15].

2.4.2 Three-valued Disjunctive Existence Results

To distinguish these problems from the above two-valued cases where all matrices contain only 0's or 1's, we adjoin an X in the notation. Here the M may contain entries from (0,1, X) where X denotes "unkown". Disjunctions are evaluated on such M by standard 3-valued logic.

I'. $D^+_{max}(X)$ is in P.

(Proof in [15].)

II' - V': The three-valued problems are NP-complete, For example, we demonstrate II'.

Proof of II': $D_{max}(X)$ is isomorphic to the originally proven NP-complete problem SATISfaction of boolean CNF's, $F = \&(C_i)$, over n atomic variables (1,2,...,n) and their negations, per the Reduction:

Suppose F has m clauses. Transform F to the mxn 3-valued matrix M where M(i,j) is: 1 if clause C_i contains variable j, 0 if C_i contains -j, and X otherwise. It can be checked that a truth assignment to (1,2,..., n) satisfying F corresponds to one set of signs on properties (columns) P1,P2,..., Pn yielding a true disjunction over M, and conversely, The map F↦M is log space.[X].

Most of the positive disjunctive forms of III' -V' follow form reductions to the NODE_COVER problem, via the previously seen NP-complete D^+par two-valued case, as shown in [15].

2.5 ASSOCIATIONAL HYPOTHESES EXISTENCE RESULTS

These problems will be denoted by "A" in place of "D", even though we still assume that the two sides of the associations are each in boolean disjunctive form: $\sim$ denotes (simple) association.

A_{max}^{+}: Given any M over (0,1), are there positive disjunctions d, d' of total length l(M) such that $d \sim d'$ is true in M?

[It is an Open Problem where the above set fits in the hierarchy!]

Likewise, $A_{max}(X)$ denotes the same problem for arbitrary disjunctive sides, but over three-valued M.

2.5.1. Two-valued Associational Results

P-time solvable: A_{max}; $A_{<}$: $A_{1/2}$.

NlogN solvable: A_{par}.

OPEN PROBLEMS: A^{+}_{par} and all such Positive cases.

2.5.2 Three-valued Associational Problems

We define here a special-form positive-parts associational problem that, as an exception, has a P-time solution. All other cognates of the above problems have been shown to be NP-complete. (See Table 2.)

$A^{+}_{max-1}(X)$: Given any M over (0,1, X), are there positive disjunctions d, d' of lengths m-1, 1 respec., such that $d \sim d'$ is true in M?

Proposition: The above problem is solvable in linear time.

Proof: The possible choices for d', hence d, are linear in l(M).[X]

The above proposition gives hope to the practical side of GUHA-style research, because almost all important (e.g., medical) questions are phrased in terms of what combination of subject properties associate with one specific property, e.g. disease. Similar considerations make disjunctive problems important to finding what implications from a conjunctive (negated premiss) combination of properties, or lack thereof, yield one highly interesting conclusion. There are as yet few results on other quantifiers, e.g. Chi-SQuare $\sim^2$. One negative result appearing in the appendix of [15] shows that all the $\sim^2$ three-valued problems are NP-complete.

Table 1. Summary of results

		Logical values			
		2		3	
	Length	$\forall$	$\sim^{\circ}$	$\forall$	$\sim^{\circ}$
±	parameter	$2^{\log^2}$	$2^{\log^2}$	*NP*-c.	*NP*-c.
	$\frac{1}{2}l(M)$	*P*	*P*	*NP*-c.	*NP*-c.
	$l(M)$	*P*	*P*	*NP*-c.	*NP*-c.
	$\leqslant l(M)$	*P*	*P*	*NP*-c.	*NP*-c.
+	parameter	*NP*-c.	?	*NP*-c.	*NP*-c.
	$\frac{1}{2}l(M)$	*NP*-c.	?	*NP*-c.	*NP*-c.
	$l(M)$	*P*	?	*P*	*NP*-c.
	$\leqslant l(M)$	*P*	?	*P*	*NP*-c.

TABLE 1. SUMMARY OF RESULTS

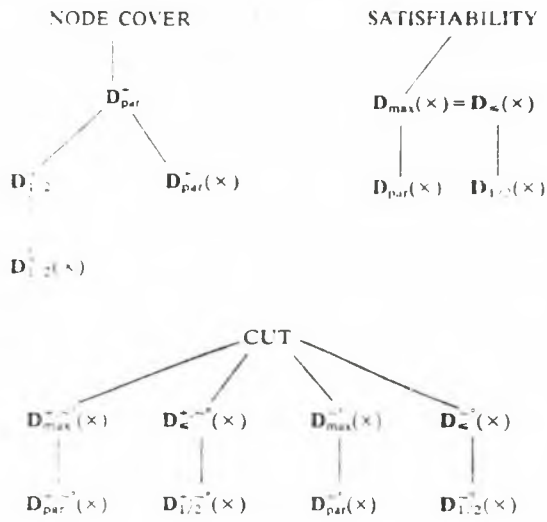


TABLE 2. TABLE OF NP-COMPLETENESS

OPEN QUESTION: Is D_{par} or A_{par} actually in P-Time? I.e., can the critical subcase for $2^k \leq m < 2^{n/2}$ be solved in time less than the exhaustion method's $n^{\log m}$? If so, such a P-time algorithm could conceivably be useful to design an MHF process to decide D_{par} for arbitrary matrices with a moderate number of rows compared to 2^n, but more than 2^k, making the Principle of Exclusion inapplicable.

GENERAL SOLUTIONS: All of the above problems have considered only existential questions, e.g. of whether there exist true length-n or -k disjunctions for the given M. This is a good prior problem to solve before running any GUHA algorithm, for if no output is to result, it would run exceedingly long to discover this!

Still, given some tractably solvable existential situation, how do we convert this to knowledge useful in generating ALL the desired, interesting hypotheses, as required by the General Goal? As seen in [16], most of these so-called "General" problems are NP-hard!

Heuristic Case I: If $2^k \leq m < 2^{n/2}$, THEN we need not generate all 2^n-m true length-n disjunctions by Exclusion; it suffices to find (some) of their minimal-length representatives, using the following heuristic, where m is approximated by 2^k:

Each length-k representative extends to 2^{n-k} length-n d's; thus the total number of the latter is represented by at most

$[2^n - 2^k]/2^{n-k} = 2^k - f < 2^k$

distince length-k true disjunctions, where $f < 1$. This assumes few short d's extend to equal length-n's, which is not always true! However, each longer one can stand for its many shorter representatives, if necessary, and at most one should generate about m "independent" disjunctions, giving a linear bound. Cf. Example A.

Heuristic Case II: IF $m < 2^k$ and $k = n/2$, as in Example B, THEN we can generate true length-k disjunctions using ANY k columns for a total possible number of (n"choose"k)*(2^k-m) length-k's, each of which extends to 2^{n-k} length-n's, for a grand product $>> 2^n - m$, the distinct disjunctions!

Thus, here again, as each length-n true disjunction may be the extension of many, many length-k's, we should use the former to keep track of the latter ones and of independence. For example, M' can have at least (6|3)= 20 length-3 truths, but each of them pairs with its complementary columns in on e of the 57 length-6 truths.

3 EXPECTED FUTURE DEVELOPMENTS IN GUHA-STYLE EDA/MHF

Future EDA/MHF software can benefit from the experience gained in developing the earlier GUHA algorithms, and in the advanced analysis of possible heuristic algorithms.

Eventually, when Automated EDA is applied to very large domains and/or data "samples", like a large city's census, it will require a CPU on the order of today's supercomputers, in order to generate all "interesting" basic rules (even of restricted forms) about the city's population. For example, many different types of economic analysis questions could be answered simultaneously, without each being explicitly asked!

3.1 WHERE WE ARE/Current limitations on automated discovery:

The major limiting factor is the high cost of discovery compared to conventional methods of doing research; specially built systems cost more than people, at least at first. However, IF vital results were discovered this way that were not seeable another way, then the value of EDA systems would be apparent, regardless of cost. It is doubtful that this can be demonstrated soon, only a relative speed of examination compared to humans.

Costs are high because:

(1) collecting and storing data is costly;
(2) building suitable (initial) KB's is expensive (EDA can help later);
(3) processing very large data sets, using the twin KB's, to find new knowledge could be very expensive (supercomputers will be justified);
(4) it is inherently costly to extract just the valid AND new hypotheses from all those generable by the system.

3.2 WHERE WE WOULD LIKE TO BE:

The G_QUANT and ASSOC systems increase our understanding of what the larger package should do, and how to do it. Also, the smaller systems could be used directly by an implementation of GUHA-80, as modules. The GUHA approach to EDA seems much in need of a normal-sized expert system for its users, partly because it is non-standard in some sense: GUHA is oriented mainly to nominal data and its procedures tend to generate plausible domain hypotheses, rather than confirming some user-posited hypotheses.

Special features of a newly proposed ("GUHA-90", below) project also make it possible for an applied EDA system to be of benefit to AI. Having large empirical data, one could process them by ASSOC with an implicational quantifier in order to obtain rules of the form:

IF (condition) THEN (conclusion) WITH CERTAINTY (c-degree).

Such rules form part of the knowledge base in most expert systems. Thus, it it conceivable that GUHA-80 could be useful to opening the bottleneck of the Knowledge Acquisition Problem that every knowledge engineer faces when building a knowledge base from the utterances of human experts. In the spirit of automated research, inspired by Tukey's exploratory data analysis, one aim of this rest-of-century project is to partially automate, and thereby speed up, the abstraction of heuristic KB rules, directly from the data as much as possible. However, this idea needs much research and testing, to test its feasibility further.

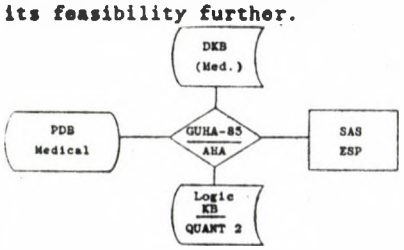


Figure 3. GUHA-85 Planned Architecture

3.3 WHERE WE CAN GO/HOW TO GET THERE: **The MOST interesting developments are expected to occur when we combine the best of both (GUHA and RX) worlds: by interfacing a domain-dependent KB with a domain-independent logico-statistical KB having the power of GUHA's general logic system, plus its multi-statistics evaluations. [See FIGURE 3.] This future system will have five major software components:**

(1) The Domain-expert KB (DKB), preferably medical for comparisons;

(2) The Logic KB (LKB), much more powerful than RX's one-on-one;

(3) The domain-dependent database (PDB), for patient studies;

(4) The statistics applications system, here called SAS; and

(5) The control system that includes an automatic hypothesis acquirer (AHA), to coordinate the parallel workings of LKB and DKB and to enlarge (occasionally) the DKB.

It seems clear that American implementations of EDA expert systems should not repeat the FORTRAN and PL/I experimental work of the Czechs. In fact, in order to handle the backward-chaining logic and rich knowledge representation framework envisaged for GUHA-80's unmet goals, several types of modern software support must be arranged:

(1) **UNIX/C for a productivity-enhancing operating environment;**

(2) **PROLOG for the AI aspects just mentioned, requiring Logic Programming;**

(3) **Compiling facilities, for calling SAS from within PROLOG;**

(4) **Test Advisor module fully integrated into the Evaluation Stat-Pak;**

(5) **Artificial Hypothesis Acquirer, transforming output hypotheses, after confirmation, into domain rules for the Knowledge Base.**

[Note that "AHA" here implies human-interfaced knowledge acquisition!]

Desirable hardware: powerful, multi-station, number-crunching super-minicomputer with massive disk storage. Medium-to-high-resolution graphics, for displays, would be an extra advantage. [While Tukey's version of EDA is more visually oriented, our system will NOT try to do cluster analysis graphically!]

AUTHOR'S NOTE: While such large research projects are inherently expensive, it seems that the most advanced nation should be able to cooperate with one of the smallest in Eastern Europe, in order to effect very state-of-the-art information "extraction" systems. Consider the gains.

(1) The RX system output is too restricted; it is domain-dependent;

(2) The GUHA system output is too prolific; it is domain-independent;

(3) The combined, binational system could use the best of both present systems: domain KB for soundness and selectivity of results, and the uniquely powerful logic KB of GUHA-85 to increase the likelihood of discovering varied, new and significant results.

ACKNOWLEDGEMENTS: I need to thank profusely my Czech friends and colleagues, mainly contacted through Dr. Petr Hájek in Prague, for the continuing inspiration that their dedication provides. Many of the complexity results herein were the work of P. Pudlák.

I also acknowledge the partial support of National Science Foundation's Information Science Program, grant IST #8503082, which currently supports my research into logical database design.

REFERENCES

1. Tukey J.W., Exploratory Data Analysis, Addison-Wesley, 1977.

2. Hajek P., Havranek T., Mechanizing Hypothesis Formation: mathematical foundations for a general theory, Springer-Verlag, 1978.

3. ---, ---, GUHA-80: an application of AI to data analysis, Computers and Artificial Intelligence 1 (1982), 107-134.

4. Hajek P., Applying AI to Data Analysis, Proc. Eurpn. Conf. on AI, Orsay France, 1982, 149-150.

5. -------, Combining functions for certainty degrees in consulting systems, Int. J. Man-Machine Studies 22 (1985), 59-76.

6. Barr A., Feigenbaum E. (eds.), The Handbook of AI, Chapter III: Knowledge Representation, pp. 141-222.

7. Lenat D., AM - an AI approach to discovery in mathematics STAN-CS-76-570, Stanford Computer Science Department 1976.

8. Dixon J. (ed.), BMDP - Biomedical Computer Programs, Univ. of Calif. Press, Los Angeles 1975.

9. Shortliffe E., Computer-based medical consultations: MYCIN, American Elsevier, New York 1976.

10. van Melle W., A domain-independent system that aids in constructing knowledge-based consulting programs, STAN-CS-80-820, Stanford Computer Science Department 1980.

11. Hart P., Duda R., Einaudi M., PROSPECTOR - a computer-based consulting system for mineral exploration, Math. Geology 10, (1978) 589-610.

12. Bennet J., Croary L., Engelmore R., Melosh R., SACON - a knowledge based consultant for structural analysis, STAN-CS-78-699, Stanford CSD 1978.

13. Hajek P., Havranek T., The new version of the GUHA-Procedure ASSOC: brief description and user's manual, Math. Inst. Tech. Rpt. 1984-#8, Czechoslovakian Academy of Sciences, Prague 1984.

14. Hajek P., The New Version of GUHA Procedure ASSOC - mathematical foundations, Proc. COMPSTAT 1984, Physica-Verlag, Vienna 1984, 360-365.

15. Pudlak P., Springsteel F., Complexity in Mechanized Hypothesis Formation, Theoretical Computer Science 8 (1979), 203-225.

16. Springsteel F., Complexity of hypothesis inference problems, Int. J. Man-Machine Studies 15 (1981), 319-332.

17. Blum R., Discovery, Confirmation and Incorporation of Causal Relationships from a large Time-oriented clinical data base, Computers and Biomedical Research 15 (1982), 164-187.

Proc.IMYCS '86 October 13-17,1986
Smolenice Castle,ČSSR

PERSPECTIVES OF LOGIC PROGRAMMING

Péter Szeredi

SzKI - Computer Research and Innovation Center
Budapest, Donàti u. 35-45. H-1015
Hungary

Abstract

In the paper we present some reflections on PROLOG and logic programming. Assuming a basic knowledge of PROLOG we try to show some aspects of both the underlying theory and the practical applications. We also try to discuss how PROLOG fits into the general stream of logic programming research reflecting on its past, present and future.

1. Introduction

Mathematical logic has been used in various areas of programming from the very beginning of its history. The basic operations of propositional calculus, the conditional statements are present in every programming language. Now we are trying to show the path of research that aims at using higher forms of mathematical logic in programming. This path leads to the creation of very high level programming languages based on logic as PROLOG is.

In point 2. some aspects of verification and program synthesis systems are discussed and the general principles of logic programming are introduced. In points 3. and 4. features of the PROLOG language are described - its relation to logic programming and also its usability in practical programming. Finally in point 5. current trends of research and development of logic programming and PROLOG are investigated.

2. From "logic in programming" to "logic programming"

Higher forms of mathematical logic, e.g. the first order predicate calculus were started to be used in programming in the early sixties. One of the research directions for facing the software crisis advocated the use of program specifications formulated in mathematical logic to provide safer and more productive software systems. On the other hand various mechanical theorem proving techniques were developed that could serve as tools for implementing such logic based systems.

The first approaches that used program specifications in logic were program verification and program synthesis systems [8]. Let us examine a much simplified schema of these systems as presented in Figs.1. and 2. In a program verification system the user has to supply a program (generally written in some high level algorithmic language) together with a specification in logic of what the given program is intended to do. The verifier uses theorem proving techniques to compare the program and the specification and returns a yes/no answer whether the program is correct with respect to the given specification or not. The process of verification can be performed independently of the actual execution of the program as depicted on Fig.1.

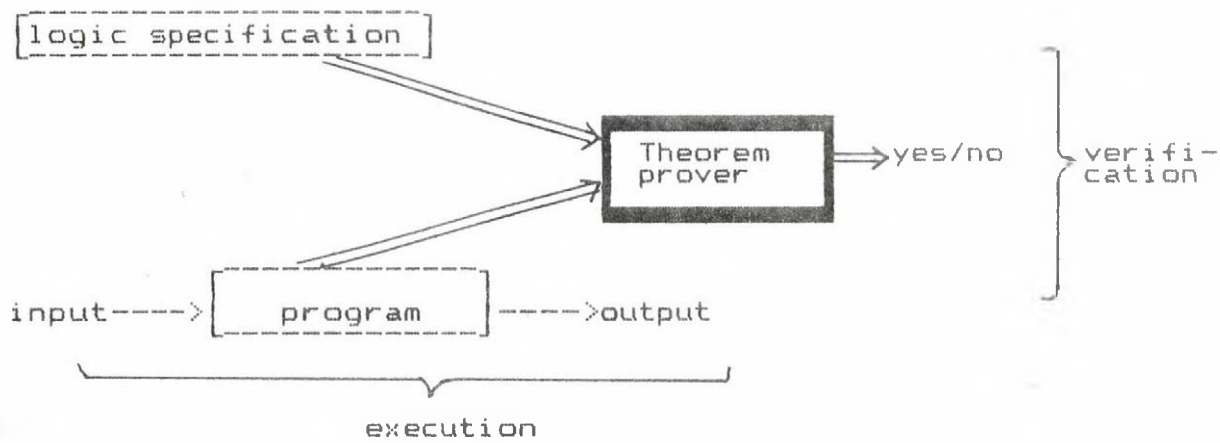


Fig.1. Schema of a program verification system

A program synthesis system takes a logic specification of a program as input and - again using theorem proving techniques - produces a program conforming to the specification (Fig.2.). Of course program synthesis needs more powerful theorem proving techniques than verification in order to construct an executable program. This program can be subsequently executed in the same way as hand written programs.

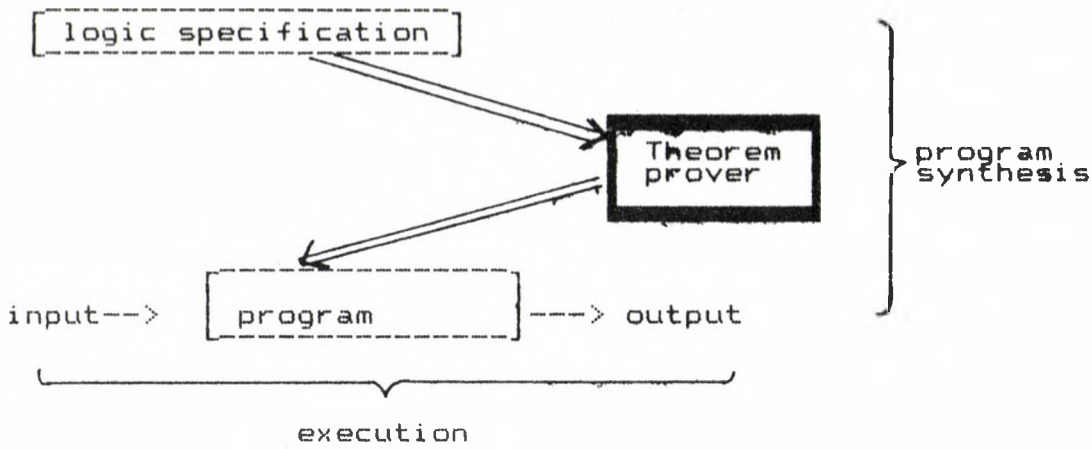


Fig.2. Schema of a program synthesis system

Experience and the problems of theorem provers, verification and synthesis systems all contributed to the appearance of a new research direction: logic programming pioneered by A.Colmerauer, P.Hayes and R.Kowalski in the early seventies ([4], [6]). The basic idea of this approach is that one can totally get rid of the "traditional" algorithmic program in the above schemas. The specification in logic itself can be considered a program that may be executed using a theorem prover, which - in terms of software engineering - serves as an interpreter for the logic program (Fig.3.):

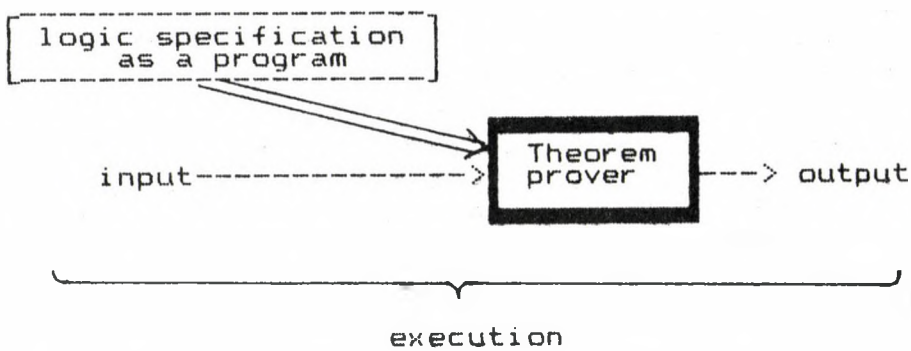


Fig.3. Schema of a logic programming system

Let us clarify the basic idea of logic programming. A logic specification can be thought of as a formula relating the input values $\underline{x}$ of a program to the output values $\underline{y}$: $\varphi(\underline{x},\underline{y})$.

If the specification is satisfiable then for each input there exists at least one set of output values, i.e

$$\forall \underline{x} \exists \underline{y} \varphi(\underline{x},\underline{y})$$

holds. (We are simplifying the discussion by assuming that the program should work for all input values. It is left to the reader to consider the case when the program and so the formula 'φ' is defined only for input values satisfying some condition '$\psi(\underline{x})$'.) Given now a concrete input $\underline{a}$ one can supply the true closed formula

$$\exists \underline{y} \varphi(\underline{a},\underline{y})$$

to a theorem prover which should be able to prove it. If the theorem proving technique is constructive, i.e. an existentially quantified formula is proved by finding a concrete object for which the formula holds, then proving the above formula means constructing a concrete $\underline{b}$ for which

$$\varphi(\underline{a},\underline{b})$$

holds. That means that the result of the theorem proving process is the desired output $\underline{b}$ of the program.

A logic programming system allows the user to write specifications instead of programs - thus implementing the famous slogan "WHAT rather than HOW". Logic programming systems, however, still inherit the major problem of theorem proving: the peril of combinatorial explosion. Proving a theorem is basically finding a specific path in a tree of possible proofs - which may be a practically impossible task if the tree is big enough. There are a number of concepts left open in the general schema of logic programming: the logic language, the theorem proving method and also the strategy of the theorem prover. An appropriate choice of these concepts can help in overcoming the problems of combinatorial explosion.

3. From logic programming to PROLOG

The programming language PROLOG (PROgramming in LOGic) was created in the early seventies at the University of Marseille [12], virtually in parallel with the construction of the theory of logic programming.

As mentioned previously some compromises had to be made in designing PROLOG to avoid combinatorial problems. The first of these was to restrict the language to so called Horn clauses, i.e. statements of form

$$(*) \qquad A \text{ if } C_1 \text{ and} \ldots \text{ and } C_n \qquad n>=0$$

where A and C_k are atomic relations having constants, variables and functional expressions as arguments. Functional expressions consist of a function name applied to a number of arguments of similar form. (*) is usually called a rule if $n>0$ and an assertion or fact if $n=0$. All variables in the statement are universally quantified. Note that for variables which occur only in conditions C_k this is equivalent to being existentially quantified whithin the conditions part:

$$A \text{ if } (\exists \underline{v}(C_1 \text{ and} \ldots \text{ and } C_n))$$

In addition to a number of statements of form (*) one can specify a goal statement

$$C_1 \text{ and } \ldots \text{ and } C_n$$

to be proved.

The theorem proving technique choosen for PROLOG is resolution developed by J.A. Robinson in 1965 [11]. This is one of the most powerful theorem proving techniques, it works on the clausal form of first order logic (of which the Horn clauses are a subset). Its basic operation is unification - a generalization of pattern matching which is used to create the least specialized common form of two atomic relation expressions by substituting variables in both of them.

A number of specialized strategies have been developed for resolution theorem proving. PROLOG uses a form of linear resolution: SL resolution [5]. In addition to that, most common PROLOG implementations apply strict selection rules in traversing the search tree of proofs. This results in a very simple proof strategy that can also be described in more traditional terms of pattern directed procedure invocation and backtracking, leading to procedural semantics which is used in most PROLOG textbooks, e.g. [2].

On approaching PROLOG from the theorem proving side one can be seriously disappointed. The restriction to Horn clauses means that one can not use negation, implication or universal quantifiers in the conditions of a rule. Consequently many relations need to be defined recursively, when one could get away without recursion in full first order logic. An example of this is the relation 'all elements of a given list are positive' which could be something like

```
all_elements_positive(L) if
            (∀ X) (member(X,L) implies X>0).
```

in full first order logic. In PROLOG one needs to transform the above to the following:

```
all_elements_positive ([]).
all_elements_positive ([X|L]) if
            X>0 and all_elements_positive(L).
```

PROLOG programs are also especially vulnerable to ordering of statements or ordering of the conditions of a rule. For example the 'ancestor_of' relation can be simply defined by a recursion on the 'parent_of' relation:

```
A ancestor_of D if A parent of D.
A ancestor_of D if A parent_of C and C ancestor_of D.
```

If we exchange the order of the two conditions in the second statement the PROLOG execution mechanism will enumerate all ancestors and then loop. If in addition we exchange the two statements then PROLOG will loop immediately.

If, however, one looks at PROLOG as a programming language it shows a number of high level features. The most important of these are the following

a. Double semantics - in addition to procedural semantics PROLOG statements have a well defined meaning in logic: a declarative semantics.

b. Multi-purpose definitions - a relation represents a number of functions depending on which arguments are supplied on invocation and which are not.

c. Backtracking can serve as a high level form of iteration.

d. Unification replaces a number of data handling facilities: selector and constructor functions and also handling of references (pointers)

Let us illustrate the above points on a classical example of list membership

```
(1) member(X,[X|L]).
(2) member(X,[Y|L]) if
                member(X,L).
```

These two statements can be read declaratively as follows:

(1) for each X and L a list with head X and tail L has X as its member i.e. a head of a list is its member;

(2) X is a member of a list if it is a member of its tail.

The definition of member can be used for a number of purposes. The simplest use is to check whether a given object is a member of a given list:

```
?- member(3,[1,2,3,4,5]).
```

The answer is: Yes.

It can be used to find a common member of two lists:

```
?- member(X,[1,2,3,4,5]) and
          member (X,[2,4,6,8,10]).
```

The system gives two answers: X=2 and X=4

Here the first call of member chooses an element of the first list while the second one checks whether the selected element is present in the second list. The second invocation of member acts as a filter on the results produced by the first one. This could be implemented by a doubly nested loop in a traditional algorithmic language. Increasing the number of conditions to be satisfied for a given object, corresponds to increasing the depth of nested loops - this shows how enumeration on backtracking in PROLOG replaces complex nested iterations of algorithmic languages.

The same definition of membership can be used to construct a list with given objects as elements. To illustrate this feature we use a more complex data structure for the elements of the list: 'pair(KEY,CONTENTS)'. A list of such pairs can be used to represent a dictionary associating a CONTENTS field with each KEY field. The 'member' predicate can be used both for entering a new pair to the dictionary and for searching a pair with a given KEY field:

```
?- member(pair(a,1),L) and member(pair(b,2),L)
                       and member(pair(b,X),L).
```

The answer is: L = [pair(a,1),pair(b,2)|L'']
X = 2

The first call of member instantiates L to [pair(a,1)|L'], the second one L' to [pair(b,2)|L''] resulting in the above answer. The third call does not instantiate L further, but returns X=2 by matching the given pair(b,X) with the second element at L. Of course, in real applications these calls of member are not consecutive, they are scattered through a recursive program. This example convincingly shows the power of unification replacing complex data manipulation constructs of algorithmic languages.

Summarizing the discussion we have to state that a PROLOG system should not be expected to behave like a general theorem prover. PROLOG should be regarded as a programming language which, however, preserves a number of important features of an ideal logic programming language. For example, in general we cannot write PROLOG programs by describing only WHAT to perform, we do have to consider HOW the program will be executed. The existence of declarative semantics means, however, that a lot of PROLOG programs can be read concentrating on WHAT the program is going to do without paying any attention to HOW this is going to be achieved.

4. From PROLOG to practice

The pure PROLOG language needs to be extended with so called built-in predicates to support "real" programming.

Introduction of some built-in predicate groups - for example of integer arithmetic predicates - do not hurt the basic principles of logic programming. In this case equivalent predicates could be formulated in PROLOG, but the underlying hardware or software can itself perform the necessary operations (e.g. addition or multiplication) so it is worthwile to allow the PROLOG programmer to access the hardware facilities through built-in predicates. Such built-in predicates can be regarded as relations that are not defined by logical formulae but derive their meaning from some standard model.

Other groups of built in predicates - basically those for input/output - need to be introduced to link the PROLOG system to the usual computing environment. Such predicates can hardly be assigned a declarative reading, their essence is the side effect they produce, but this side effect concerns the external world not the PROLOG system itself.

There are two built-in predicate groups which may cause conflicts with declarative semantics and so give rise to the bulk of controversy: the predicates for controlling the execution and the predicates for program modification (or PROLOG data base handling).

The basic control operation available in all PROLOG implementations is the 'cut' operation denoted by '!' (or sometimes by '/'). One can use the 'cut' just to improve

efficiency of the program i.e. to cut out only those branches of the search tree that are known to contain no solutions. This use does not modify the semantics of the program, so the 'cut' can be ignored on declarative reading. The "real" use of the 'cut' is when those branches are cut out which may contain solutions. This is a serious breach of declarative semantics, which, however, can be remedied in certain situations. For example one can give declarative semantics to certain constructs in which 'cut' occurs (switching to a more widespread notation for PROLOG connectives ':-' and ',' instead of if and and):

```
d :-
   a, !, b.
d :-
   c.
```

can be read as

d if (if a then b else c)

or equivalently

d if (a implies b and not(a) implies c)

provided 'a' is fully instantiated (does not contain free variables) at the moment of execution and also accepting the 'closed world assumption' i.e. that 'not(a)' can be considered true if the proof of 'a' fails.

The program modification predicates are for adding and deleting statements. This of course carries the danger of self modifying programs. The most common use of these built-in predicates is, however, for updating a database of variable-free facts. Even this simple usage is difficult to embed into the theory of logic programming since the modification of a database of facts means changing the axioms during the proof. Let us show a small extension of the "standard" PROLOG predicate set that helps in solving the outlined problem.

In the MPROLOG system ([1] [9]) backtrackable versions of database modification predicates are introduced. With these predicates the change performed on the database is undone when control backtracks over the predicate causing the change. This means that if several alternative branches can be selected at a

given point of execution the data base will be restored to the original state before trying a next alternative.

The backtrackable versions of the data base modification predicates are much more clear from the theoretical point of view than the non-backtrackable ones. In fact, the solution proposed by R. Kowalski [7] to overcome the impurity introduced by a changing data base through meta-level reasoning works only for the backtrackable version. On the other hand backtrackable predicates are very useful in the practice of PROLOG programming. We illustrate this by a small example in MPROLOG.

Let us have an undirected graph represented by a data base of facts of form

edge(point$_1$,point$_2$)

Such a fact states that there is an edge from 'point$_1$' to 'point$_2$'. We define a predicate 'path(point$_1$,point$_2$) to mean there is a path from 'point$_1$' to 'point$_2$':

```
path(X,X).
path(X,Z):-
     del_matching_edge(X,Y),path(Y,Z).

del_matching_edge(X,Y):-
     del_matching_statement_b(edge(X,Y)).
del_matching_edge(X,Y):-
     del_matching_statement_b(edge(Y,X)).
```

Here del_matching_statement_b is the backtrackable predicate to find a statement matching its argument and to delete it. Deletion of the matching fact ensures that each edge is used only once, but if backtracking steps back over the given choice the deleted fact has to be reinserted into the data base to make possible using the given edge in another selection. Note that in the above example 'del_matching_statement_b' could be defined in terms of del_matching_statement (equivalent of 'retract' in DEC-10 PROLOG) and add_statement (equivalent of 'assert'):

```
del_matching_statement_b (ST):-
          del_matching_statement(ST).
del_matching_statement_b (ST):-
          add_statement(ST), fail.
```

This definition, however, is not only more expensive in terms of implementation, but it is also vulnerable to any 'cut's - these may cut off the 'add_statement' branch in which case undoing is not performed.

5. Future trends in logic programming

The first, very natural question arising in connection with the future of logic programming is: can one expect new logic programming languages to be created that are completely different from PROLOG?

Recent results of M. Szöts [13] show that a constructive theorem proving technique can be developed only for sublanguages of logic that are equivalent to (some subset of) Horn clauses. By constructivity we mean here that at proving an existentially quantified formula a unique object is being constructed for which the formula holds. This result proves that the language of a logic programming system (as outlined in point 2) must be equivalent or weaker than the language of Horn clauses, which means that the subset of logic chosen for PROLOG was in some sense the "best choice". On the other hand we can still create new logic programming languages as long as the language is transformable to Horn clauses in general terms of logical equivalence. In this sense PROLOG can be regarded as a low level, "machine code" language of logic programming and one can design "higher level" logic programming languages that can be transformed to Horn clauses - just as higher level algorithmic programming languages are transformed into actual machine code.

One important aspect where PROLOG needs to be upgraded to a higher level is organisation of loops or more exactly recursion. The restriction to Horn-clauses means that one has to use recursion instead of universal quantification as illustrated in point 3 by the 'all_elements_positive' example. Moreover many PROLOG definitions have to be defined by recursion, e.g. the 'split' predicate used in quicksort:

```
split([H|L],X,[H|L1],L2):-
     H<=X, split(L,X,L1,L2).

split([H|L],X,L1,[H|L2]):-
     H>X, split(L,X,L1,L2).

split([],_,[],[]).
```

could be defined in a "higher level" form by something like

```
split(L,X,L1,L2) if
     L1 = list(H elem L suchthat H <= X) and
     L2 = list(H elem L suchthat H > X).
```

Another possible extension of the PROLOG language is the introduction of data types. This helps in improving the readability of programs and also may contribute to solving "loop" organization problems. A recursive data structure e.g. a binary tree can be traversed in several ways - so the language should enable the user to define the data structures and the traversal algorithms together. The traversal algorithms could be named (e.g. left_to_right_breadth_first) and used in loop (recursion) specifications.

Introduction of data types may also help in improving the speed of code generated from a PROLOG program by a compiler. It is a problem, however, that currently available PROLOG systems offering data types (e.g. TURBO-PROLOG based on work of J.F. Nilsson [10]) pose severe restrictions on important aspects of PROLOG e.g. data base handling and meta-programming.

Let us now briefly tackle the two other components of a logic programming system besides the language: the theorem proving technique and the strategy (execution mechanism). Resolution and especially unification seem to have important advantages over other techniques. An example of very few alternative languages is "LOBO" [3]. "LOBO" has no pattern matching but it allows bounded universal quantifiers resulting in a logic programming language that is nearer to traditional programming and simpler to compile - but one looses quite a few advantages of PROLOG, e.g. those described under b. and d. in point 3.

Many research activities have been devoted to improving the very simple and straightforward execution mechanism of PROLOG. Due to lack of space we just list a few topics:

- intelligent backtracking tries to avoid those branches of the search tree which can not change the subgoal that causes backtracking.

- coroutined execution to allow postponing certain subgoals until e.g. some variables become instantiated.

- concurrent execution that makes possible exploiting parallel hardware.

6. Conclusion

PROLOG has been a compromise between the general aims of logic programming on one side and the capabilities of hardware and the power of implementation techniques on the other side. Recently there have been important developments on this second side: dramatic increase of the computing power of generally available hardware and also notable improvements in the compilation technique of PROLOG (see e.g. [10] and [14]).

This will hopefully contribute both to more widespread use of PROLOG which may become almost a general purpose language like Pascal and also to the development of higher level logic programming languages that may bring the "WHAT rather than HOW" principle nearer to realisation.

References

[1] J.Bendl, P.Koves and P.Szeredi: The MPROLOG System; Proceedings of the Logic Programming Workshop, Debrecen, 1980, pp. 201-209.

[2] W.F.Clocksin and C.S.Mellish: Programming in Prolog, Springer Verlag, 1981.

[3] T.Gergely and M.Szöts: Some features of a new logic programming language; Proc. of Workshop and Conference on Applied AI and Knowledge Based Expert Systems, Ed. P.Revai Univ.of Stockholm, 1985.

[4] P.J.Hayes: Computation and Deduction; Proceedings of Symposium on Mathematical Foundation of Computer Science, High Tatras September 1973, pp.105-117.

[5] R.A.Kowalski and D.Kuehner: Linear resolution with selection function; Artificial Intelligence 2, 1971, pp. 227-260.

[6] R.A.Kowalski: Predicate Logic as Programming Language; Proc. IFIP'74, North Holland Publ. Co., Amsterdam, 1974, pp. 569-574.

[7] R.A.Kowalski: Logic Programming; Proc. IFIP'83 North Holland Publ. Co., Amsterdam, 1983, pp. 133-145.

[8] Z. Manna: Mathematical Theory of Computation; McGraw Hill, New-York, 1974.

[9] MPROLOG Documentation, Release 2.1; Logicware, Toronto; SzKI, Budapest, 1985.

[10] J.F.Nilsson: On the compilation of a Domain Based Prolog; Proc. IFIP'83, North Holland Publ. Co., Amsterdam, 1983, pp. 293-298.

[11] J.A.Robinson: A machine-oriented Logic Based on the Resolution Principle; Journal of ACM 12, pp. 23-41.

[12] Ph.Roussel: PROLOG: Manuel de Reference et d'Utilisation; Grouppe d'Intelligence Artificielle; Universitè d'Aix-Marseille,Luminy, 1975.

[13] M.Szöts: Logical Foundations of Logic Programming; Thesis, Budapest, 1986.

[14] D.H.D.Warren: An Abstract Prolog Instruction Set; Technical Note 309, SRI International, Oct. 1983.

SHORT COMMUNICATIONS

Proc. IMYCS '86 October 13-17, 1986
Smolenice Castle, ČSSR

NONDETERMINISM IS ESSENTIAL FOR REVERSAL - BOUNDED TWO - WAY MULTIHEAD FINITE AUTOMATA

Andrej Bebják, Ivana Štefánekvá

Department of Theoretical Cybernetics
Comenius University
842 15 Bratislava
Czechoslovakia

1. Introduction

The question whether nondeterminism is more powerful than determinism is one of the most investigated questions in complexity theory. The well-known extentions of this question are NP ? P, DLOG ? NLOG. We shall study the second in this paper.

We shall consider two-way deterministic /nondeterministic/ automata, 2dfa(k) /2nfa(k)/, because they characterise logarithmic space in the following way

$$DLOG = \bigcup_{k\in N} 2DFA(k) \ , \quad NLOG = \bigcup_{k\in N} 2NFA(k),$$

where 2DFA(k) /2NFA(k)/ is the family of languages recognized by 2dfa(k) /2nfa(k)/ automata.

Hromkovič proved in [4] that a specific language can be recognized by no 2nfa(k) automaton /for any k∈N/ with n^a, for $0 < a < 1/3$, bound on the number of head reversals in the accepting computations. Using this fact we prove that, for some "nice" functions f, cf(n) reversal-bounded 2nfa(k) automata are not closed under complement, and that cf(n) reversal-bounded 2dfa(k) automata are closed under complement, where c∈N. This separates nondeterminism from determinism for reversal-bounded multihead finite automata. The immediate consequence of this fact is that if deterministic multihead

finite automata are as powerful as nondeterministic ones then the deterministic automata have to use a substantially larger number of reversals than nondeterministic automata.

This paper is divided as follows. Section 2 involves some basic definitions and Section 3 contains the basic results concerning "f(n) reversal computability" for multihead finite automata introduced in Section 2. The main results are formulated in Section 4.

2. Definitions

The formal definition of two-way multihead finite automata as a 5-tuple (Σ,K,F,δ,q) can be found in [6], and was soon thereafter extensively studied by Sudborough [7], Yao and Rivest [8], and Hromkovič [3].

A bound on the number of reversals as a measure of complexity was introduced by Hartmanis [2]. He considered two-tape Turing machines with one-way read-only input-tape. Reversal-bounded one-tape Turing machines were considered in [1] and reversal-bounded multi-tape Turing machines in [5].

Let f be a function from natural numbers to the positive real numbers, and let M be a family of languages recognized by multihead finite automata from an automaton class S. Then M-R(f) is the class of languages accepted by automata in S which use in their accepting computations at most cf(n)head reversals for input words of lenght n.

Definition 2.1. Let f:N→N call a reversal computable function if there exists 2dfa(k)-R(cf) automaton A where c is a positive constant such, that after finishing computation on the input word of lenght n the positions of the heads on the input tape represent value f(n).

In what follows we shall only consider functions for which $f(n) \leq n$. For the representation of value f(n) we shall use head H_1, whose position on i-th symbol represents value i. When proving qualities of the operation O for $f_1,\ldots,f_m$ the result of which is f_{m+1} we suppose that $f_1,\ldots,f_m,f_{m+1} \leq n$.

3. Reversal computable functions

In this Section we shall prove that sum, integer-valued power and root of reversal computable functions are reversal computable functions.

Lemma 3.1. For every $m \in N$, if $f_1,\ldots,f_m$ are reversal computable functions then $f_1+\ldots+f_m$ is a reversal computable function too.

Lemma 3.2. If f is a reversal computable function, then f^i, where $i \in N$, is a reversal computable function.

Proof. By induction according to i. For i=1 the statement is obvious. Let us assume that f^{i-1} is reversal computable, i.e. there is an automaton A_1 whose computation ends in such a way, that H_1^1 represents $f^{i-1}(n)$ and the number of reversals is $O(f^{i-1}(n))$. Let automaton A_2 constructs f(n) so, that H_1^2 represents f(n). Automaton A working in three phases constructs $f^i(n)$ as follows:

1. It simulates A_1, resulting in H_1^1 representing $f^{i-1}(n)$, number of reversals being $O(f^{i-1}(n))$.

2. It simulates A_2 resulting in H_1^2 representing f(n), number of reversals being $O(f(n))$.

3. It moves head H_1 f(n) times by $f^{i-1}(n)$ symbols in the following way. Head H_1 will at the end represent $f^i(n)$. Heads H_1^1 and G will alternatively represent $f^{i-1}(n)$. Head H_1^1 is moved to the left and heads H_1, G to the right until H_1^1 reads ¢. Head H_1^2 is moved one symbol to the left because H_1 was moved once by $f^{i-1}(n)$. Now positions of G and H_1^1 are alternatively exchanged moving H_1^2 by one symbol to the left at every exchange. It is repeated until H_1^2 reads ¢. Number of reversals is $2.f(n)+1$.

It follows from the description of the computation of A that $f^i(n)$ is reversal computable.

Consequence 3.3. If f(n) is a reversal computable function with $O(f(n))$ number of reversals, then $f^i(n)$, where $i \in N$, is reversal computable with $O(f(n))$ number of reversals.

Lemma 3.4. $\lfloor n^{1/i} \rfloor$ is a reversal computable function, $i \in N$.

Proof. We construct 2dfa(k)-R($\lfloor n^{1/i} \rfloor$) automaton A which gradually verify if $1^i \stackrel{?}{=} n$, $2^i \stackrel{?}{=} n$ etc. If for number j holds $j^i \leq n$ and $(j+1)^i > n$, then $j=\lfloor n^{1/i} \rfloor$. Automaton A use heads G, $H_1, H_2, \ldots, H_i, \ldots, H_k$. Head G represents by its position gradually values 1^i, 2^i, 3^i... until reads %. Head H_1 moves after each successful cycle /i.e. transition from configuration in which head G represents value x^i to configuration in which head G represents value $(x+1)^i$/ one symbol to the right and at the end represents value $\lfloor n^{1/i} \rfloor$. On transition from x^i to $(x+1)^i$ heads $H_2, \ldots, H_i$ always represent following informations: H_2 represents value $\binom{i}{1}x^{i-1}$, H_3 represents $\binom{i}{2}x^{i-2}$, .., H_i represents $\binom{i}{i-1}x$. According to binomial theorem $(x+1)^i = x^i + \binom{i}{1}x^{i-1} + \ldots + \binom{i}{i-1}x + 1$ and so it is sufficient /if possible/ to move gradually head G to the right by values represented by heads $H_2, \ldots, H_i$ and eventually move it one more symbol. Each of the heads $H_2, \ldots, H_i$ has it own substitute which in the case of value addition keeps information represented by head. Number of heads is constant and every addition requires a constant number of head reversals. Automaton must then adapt information carried by heads $H_2, \ldots, H_i$ to values $\binom{i}{1}(x+1)^{i-1}$, $\binom{i}{2}(x+1)^{i-2}$, ... , $\binom{i}{i-1}(x+1)$

$$\binom{i}{1}(x+1)^{i-1} = \binom{i}{1}\left(x^{i-1} + \binom{i-1}{1}x^{i-2} + \binom{i-1}{2}x^{i-3} + \ldots + \binom{i-1}{i-2}x + 1\right)$$

$$= \underbrace{\binom{i}{1}x^{i-1}}_{H_2} + \underbrace{\binom{i}{1}\binom{i-1}{1}x^{i-2}}_{2 \times H_3} + \underbrace{\binom{i}{1}\binom{i-1}{2}x^{i-3}}_{3 \times H_4} + \ldots$$

$$\binom{i}{2}(x+1)^{i-2} = \underbrace{\binom{i}{2}x^{i-2}}_{H_3} + \underbrace{\binom{i}{2}\binom{i-2}{1}x^{i-3}}_{3 \times H_4} + \underbrace{\binom{i}{2}\binom{i-2}{2}x^{i-4}}_{6 \times H_5} + \ldots$$

etc.

The whole adaption of information at every transition from x^i to $(x+1)^i$ always requires constant number of reversals. Since maximum successful cycles is $\lfloor n^{1/i} \rfloor$, so the number of reversals is $O(\lfloor n^{1/i} \rfloor)$.

Consequence 3.5. Function $\lfloor n^{1/q} \rfloor^p$, for $p,q \in N$, $p \leq q$, is reversal computable.

Lemma 3.6. Function $\lfloor \log_2 n \rfloor$ is reversal computable.

4. Determinism versus nondeterminism for reversal-bounded multihead finite automata

Now, proving that nondeterminism is essential for reversal-bounded two-way multihead finite automata we give a partial answer to the well-known open problem whether nondeterministic multihead finite automata are more powerful than deterministic ones.

Theorem 4.1. If a function f(n) is reversal computable, then the class of languages $\bigcup_{k\in N}$ 2DFA(k)-R(f) is closed under complement.

Proof. It is sufficient to prove that for every language L in $\bigcup_{k\in N}$ 2DFA(k)-R(f) recognized by a 2dfa(k)-R(f) automaton $A = (\Sigma, K, F, \delta, q_0)$ there is a 2dfa(k')-R(f) automaton A' $A' = (\Sigma, K', F', \delta', q_0')$ recognizing language L^c. Automaton A' works on the input word w of lenght n as follows:

In the first phase value f(n) is coded by the position of one of its heads. Let us call this head G. To code this value at most $c_1.f(n)$ reversals are necessary since f(n) is a reversal computable function. Then A' moves to the initial state q_0 of the automaton A.

In the second phase of the computation A' simulates computation of the automaton A. For every reversal of one of the heads of the automaton A automaton A' moves the head G one symbol to the left. During the computation only one of the following three cases is possible:

1. Automaton A would enter the state $q \in F$. Then A' does not accept and enters a state $p \notin F'$.
2. If head G of the automaton A' reads ¢, then the automaton A would already cross bound of reversals and therefore automaton A' enters a state $p \in F'$.
3. The head G of the automaton A' does not read yet ¢ and for state p, in which automaton A would be, δ-function is not

defined. In such case automaton A´ accepts.
Since we simulated the work of the automaton A, the number of reversals does not /even in this part of computation of A´/ exceed the value $c_2.f(n)$, where c_2 is a positive constant. Hence A´ is 2dfa(k´)-R(f) automaton and recognizes L^c.

Theorem 4.2. The class of languages $\bigcup_{k\in N}$ 2NFA(k)-R(n^a), where $0<a<1/3$, is not closed under complement.

Proof. Let us consider the language L, for which it is proved in the paper [4], that L does not belong to $\bigcup_{k\in N}$ 2NFA(k)-R(n^a), where $0<a<1/3$. It is sufficient to prove, that the complement ofthe language $L = \{ x_1\#x_2\#\ldots x_k\# \mid k \geq 0,\ x_i \in L_R \text{ for } i=1,2,..,k\}$, where $L_R = \bigcup_{r\in N} L_r$,
$L_r = \{ w_1cw_2c\ldots cw_r+w_rc\ldots cw_2cw_1 \mid w_i \in \{0,1\}^* \text{ for } i=1,\ldots,r\}$
is recognized by a 2nfa(2)-R(2) automaton A.
Let us analyze in more detail, what is the stucture of the words of the language L^c. Since every word in L /besides ε / contains symbol # , is $W_1 = \{0,1,c,+\}^+ \subset L^c$. Language consists further of all words from W_2 and W_3
$W_2 = \{ w \mid w = \#v,\ v \in \{0,1,c,+,\#\}^* \}$,
$W_3 = \{ w \mid w = vx,\ v\in\{0,1,c,+,\#\}^* ,\ x\in\{0,1,c,+\}\}$.
All other words of language L^c /i.e. those which do belong neither to W_1, W_2 nor to W_3/ have following structure:
$w = x_1\#x_2\#\ldots\#x_k\#$, where $k\geq 1$, $x_i \in \{0,1,c,+\}^*$ for i=1,..,k, $x_1 \neq \varepsilon$ and there is i such that $x_i \notin L_R$.
Automaton A nondeterministicly decides at the beginning of the computation on word u, which one of the four above mentioned structures is aquiered by the word u. To verify the first three cases one head is sufficient and this one executes not even one reversals. In the fourth case the automaton A again nondeterministicly decides which one of the subwords x_i does not belong to L_R. To verify its decision it is sufficient, for the automaton, to use two heads which execute two reversals.

As a consequence of Theorem 4.1. and Theorem 4.2. we have

Theorem 4.3. For any reversal computable function f such that $f(n) \leq n^a$, where $0 < a < 1/3$

$$\bigcup_{k \in N} 2DFA(k)\text{-}R(f) \subsetneq \bigcup_{k \in N} 2NFA(k)\text{-}R(f)$$

Corollary 4.4. For any $p \in N$

$$\bigcup_{k \in N} 2DFA(k)\text{-}R(\lfloor \log_2 n \rfloor^p) \subsetneq \bigcup_{k \in N} 2NFA(k)\text{-}R(\lfloor \log_2 n \rfloor^p)$$

Corollary 4.5. For $p,q \in N$, $0 < p/q < 1/3$

$$\bigcup_{k \in N} 2DFA(k)\text{-}R(\lfloor n^{1/q} \rfloor^p) \subsetneq \bigcup_{k \in N} 2NFA(k)\text{-}R(\lfloor n^{1/q} \rfloor^p)$$

References

1. Fischer, P.C.: The reduction of tape reversals for off-line one-tape Turing machines. J. Comput. System Sci. 2, (1968) 136-147
2. Hartmanis, J.: Tape reversal bounded Turing machines computations. J. Comput. System Sci. 19, (1979) 145-162
3. Hromkovič, J.: One-way multihead deterministic finite automata. Acta informatica 19, (1983) 377-384
4. Hromkovič, J.: Fooling a two-way nondeterministic multihead automaton with reversal number restriction. Acta Informatica 22, (1985) 589-594
5. Kameda, T., Vollmar, R.: Note on tape-reversal complexity of languages. Inform. Control 17, (1970) 203-215
6. Piatkowski, T.F.: N - head finite state machines. Ph.D. Dissertation. University of Michigan, (1963)
7. Sudborough, I.H.: One-way multihead writing finite automata. Inform. Control 30, (1976) 1-20
8. Yao, A.C., Rivest, R.L.: K+1 heads are better than K. J. ACM 25, (1978) 337-340

Proc. IMYCS '86 October 13-17, 1986
Smolenice Castle, ČSSR

ON DEPENDENCIES FOR HIERARCHICAL DATA STRUCTURES

K. Benecke

Section Mathematik/Physik
Technical University "Otto von Guericke"
3010 Magdeburg, PSF 124
DDR

1. Introduction - Motivation

We noticed that a lot of queries cannot be or cannot be conveniently expressed in terms of the Relational Algebra or even in terms of "userfriendly" languages based on the Relational Data Model. There are several attempts to avoid the lacks of the Relational Data Model. By some attempts it is tried to generalize and extend the theoretical base of the Relational Data Model, proposed by E.F. Codd, and by other attempts a completely new way is taken.
Database Logic by B.E. Jacobs /5/ is a theoretical base containing the Relational Calculus as a special case. Here, flat relational tables, hierarchical tables, and network tables are allowed. Concepts, based on Database Logic are for example "A generalized Query-by-Example Data Manipulation Language" of Jacobs and Walczak /6/ or the "Operators for Non-First-Normal-Form relations" of Fischer and Thomas /4/. Although both concepts have the same theoretical base they have nearly nothing in common at external level - the point of view of users and especially endusers. Ideas connected with the "Universal Relation" belong to the first category, too. Whereas the Functional Data Model is completely independent of the Relational Data Model Shipman /9/ or Zlatuska /10/.
The "Algebraic Founded Hierarchical Data Model" aims to provide operations for the manipulation of mass data. Its

theoretical background is an Algebraic Specitication Language for Abstract Data Types (compare Reichel /7/). The specification language forces the designer to think about each special case of an operation, to think about the generation of the basic objects and about the basic properties of operations, which determine the operations uniquely. Further, the objects and operations are freed from details of concrete representation. The design process of the operations of the data model differs deeply from the theories mentioned above.
We did not try to design basic operations as simple as possible to guarantee that it could be understood by endusers. In our belief, there is no chance to design very simple operations with the help of which the enduser is able to formulate nearly all of his query or update requirements.
We believe that a small number of universal and natural principles is needed. By the understanding of only these principles the user has to be enabled already to formulate his queries.
Some of these principles are in short:

- each table has a head, by which the columns and the structuring of the informations are described;
- a query is formalized by a GIB-AUS-MIT (GET-FROM-WHERE) construct, where
  - the head of the desired structure is written after GIB;
  - the tables (permanent files) needed for the query are written in the AUS part; they are "joined" to the source structure;
  - by conditions the desired parts of the source structure are described.

Now, it is on the designers turn to define and specify operations, by which the above GIB-AUS-MIT-construct is precised. Especially, it is necessary to design an operation, by which an arbitrary given structure can be transformed to an arbitrary other table, from which only the head is given. This operation is called stroke-list-operation or shortly stroke, because the transformation of tables is combined with aggregations, which are executed in the case of "COUNT" "stroke

by stroke".
The source structure composition is mainly realized by the extension operation (ext). We found out that the join of Relational Algebra is for many cases not the best way for putting two relations together.
Let us consider a standard example with employee and department data: EMP: S(ENO,NAME,DNO) ; DEPT: S(DNO,MANAGER_NO) (Here "S" stands for "set"). The result of the extension of DEPT by EMP is a non-first-normal-form relation with head S(DNO,MANAGER_NO,S(ENO,NAME)), in which each DEPT-pair is extended by the set of the corresponding employees. The advantages of such a non-flat structure compared with the join of DEPT and EMP are additional selecting possibilities, no undesired loss of informations, and naturalness.
On the other hand it is disadvantageously that the scheme S(ENO,NAME,DNO,S(MANAGER_NO)) is the head of ext(EMP,DEPT). Here, the inner collection of each employee is a singleton. To improve the ext operation by omitting of unnecessary structuring, it is necessary to have some 'knowledge' about the given tables. From the above example, we "know" that DNO is the key of DEPT and that each DNO-value of EMP occurs in DEPT, too. The improved operation is called "database extension" (dbext). It is an intelligent database operation. Here, "intelligent" means that the operation exploits not only the simple knowledge of heads of tables but some integry constraints, which are presupposed to hold in the given tables. The constraints represent knowledge about the values of the tables.
We see that data dependencies are a corner stone for the further development of the data model.
In this paper inference rules for Uniqueness, Existential, and Functional dependencies are proved. Such rules have to be applied before the intelligent database operations can be executed. It is outside of the scope of this paper that all basic objects and operations needed for the introduction of dependencies are specified in detail. They will be verbally explained.

3. Data Dependencies for Hierarchical Structures

In this chapter we describe shortly the objects and operations needed for Functional Dependencies. Axioms are omitted. Values of hierarchical structures are restricted to natural numbers. For more details compare Benecke /1/ and /3/.

sorts Nat, Bool (natural numbers, truth values)
sorts Fields (parameter for column names)
Coll-sym (sort for the three collection symbols)
opers set; ms; seq ---> Coll-sym (set; multiset(bag);sequen.)
sorts Scheme (head of a hierarchical table)
opers empty-s ---> Scheme (the empty scheme)
(Field) ---> Scheme (injection; without name)
coll(Coll-sym,Scheme) ---> Scheme (putting a collection symbol on the top of the scheme)
pair-s(Scheme,Scheme) ---> Scheme (concatenation of schemes)
coll?(Scheme) ---> Bool (Is the scheme a collection?)
coll-type(s:Scheme iff coll?(s)=true) ---> Coll-sym (the uppermost collection symbol of s)
red(s:Scheme iff coll?(s)=true) ---> Scheme (s is reduced by coll-type(s))
sorts Table (hierarchical table)
opers abs-empty ---> Table (absolute empty table)
empty(s:Scheme iff coll?(s)=true) ---> Table (empty collection table)
ele-tab(f:Field, n:Nat) ---> Table (elementary table)
head(Table) ---> Scheme (head of a table)
add(t1:Table,t2:Table iff red(head(t1))=head(t2)) ---> Table (adding the "element" t2 to the collection t1)
pair(Table,Table) ---> Table (putting two tables horizontally together)
comp?,inn-comp?(s1:Scheme,s2:Scheme) ---> Bool (each component of s1 is a component resp. inner component of s2)
disjoint?(Scheme) ---> Bool (the scheme contains no field multiple)
narrow?(Scheme) ---> Bool (the scheme does not contain

two collection schemes on one horizontal level; further the scheme is disjoint and does not contain an inner component of type "coll(cs,empty-s)")

ele-comps, non-ele-comp-s(Scheme) ---→ Scheme (the fields resp. collections of a scheme)

stroke(s:Scheme, p:Scheme,t:Table iff ele-comp-s(s)=empty-s & elecomp-s(p)=empty-s) ---→ Table
(the source table t is transformed element by element into a new table with head s ; p is a parameter describing the atomic collection; the atomic collections of t are not transformed element by element but as a indestructible unit; for our purposes aggregations are not needed)

A more detailed description is presented in /1/.

Definition of dependecies in hierarchical structures.

Let t be a table with narrow head and let s and s' be two narrow schemes satisfying the following property:

If c and c' are components of s and s', respectively, (*) with a common field, then c and c' are equal.

Further, let tt=stroke(S(s,S(s')),non-ele-comp-s(pair-s(s,s')),t).

The existential dependency (ED) s --e--→ s' holds in t, if each inner collection of tt with head S(s') contains at least one element;

the uniqueness dependency (UD) s ===⇒ s' holds in t, if each inner collection of tt with head S(s') contains at most one element;

the functional dependency (FD) s ---→ s' holds in t, if the ED s --e--→ s' and the UD s ===⇒ s' hold in t.

3. Inference Rules

In this chapter we shall formulate and prove the reflexivity (Ref), transitivity (Trans), projectivity (Proj), augmentation (Aug), and additivity (Add) rules for existential (e), uniqueness (u), and functional (f) dependencies. It is presupposed in any rule that the left and right hand side of the resulting dependency are narrow schemes

satisfying the above condition (*).

Ref-e

$s \xrightarrow{e} s$

Ref-u

$s \Longrightarrow s$

Ref-f

$s \longrightarrow s$

Trans-e

$$\frac{s \xrightarrow{e} s' \quad s' \xrightarrow{e} s'' \quad s' \Longrightarrow s}{s \xrightarrow{e} s''}$$

Trans-u

$$\frac{s \Longrightarrow s' \quad s' \Longrightarrow s'' \quad s\ s'' \xrightarrow{e} s'}{s \Longrightarrow s''}$$

Trans-f

$$\frac{s \longrightarrow s' \quad s' \longrightarrow s'' \quad s\ s' \xrightarrow{e} s'' \quad s\ s'' \xrightarrow{e} s'}{s \longrightarrow s''}$$

Proj-e

$$\frac{s \xrightarrow{e} s' \quad comp?(s'',s')}{s \xrightarrow{e} s''}$$

Proj-u

$$\frac{s \Longrightarrow s' \quad comp?(s'',s') \quad s\ s'' \xrightarrow{e} s'}{s \Longrightarrow s''}$$

Proj-f

$$\frac{s \longrightarrow s' \quad comp?(s'',s') \quad s\ s'' \xrightarrow{e} s'}{s \longrightarrow s''}$$

Aug-e

$$\frac{s \xrightarrow{e} s' \quad comp?(s,s'') \quad s \Longrightarrow s''}{s'' \xrightarrow{e} s'}$$

Aug-u

$$\frac{s \Longrightarrow s' \quad comp?(s,s'')}{s'' \Longrightarrow s'}$$

Aug-f

$$\frac{s \longrightarrow s' \quad comp?(s,s'') \quad s \Longrightarrow s''}{s'' \longrightarrow s'}$$

Add-e

$$\frac{s \xrightarrow{e} s' \quad s \xrightarrow{e} s''}{s \xrightarrow{e} s'\ s''}$$

Add-u

$$\frac{s \Longrightarrow s' \quad s \Longrightarrow s''}{s \Longrightarrow s'\ s''}$$

Add-f

$$\frac{s \longrightarrow s' \quad s \longrightarrow s''}{s \longrightarrow s'\ s''}$$

First, it can be seen easily that the rules Ref-f, Proj-f, Aug-f, and Add-f are immediate consequences of the corresponding existential and uniqueness rules. Further, the rule Proj-u can be derived from Ref-u, Aug-u, and Trans-u. Namely, $s \Longrightarrow s'$, $comp?(s'',s')$, $s\ s'' \xrightarrow{e} s'$ implies successively $s'' \Longrightarrow s$, $s' \Longrightarrow s$, and $s \Longrightarrow s''$. In the same way Aug-e is implied by Ref-e, Proj-e, and Trans-e. We remark that the rules

$$\frac{comp?(s,s')}{s' \longrightarrow s} \qquad \text{and} \qquad \frac{s \xrightarrow{e} s' \quad s\ s' \xrightarrow{e} s''}{s \xrightarrow{e} s''}$$

can be derived using the above rules.

Now, it is no problem to derive Trans-f.
It is evident that s --e--> s and s ===> s hold in an arbitrary table. The proofs of the rules Trans-e, Trans-u, Proj-e, Aug-u, and Add-u and Add-e are similar; only the first two are executed.
Let t be a table with narrow head. In the following, an s-element out of t is understood to be a table with head s such that its components occur in t and for each two components c and c' either c is superordinated to c' or c' is superordinated to c (or c and c' are at one level).

Trans-e

Let e be an s-element out of t. s --e--> s' implies the existence of e' such that e e' is an s s'-element out of t and s' --e--> s" implies the existence of an s' s"-element e' e" out of t. If all components of s" are superordinated to the deepest components of s and s', the e e' can be immediately extended to an s s' s"-element e e' f". Otherwise, a deepest component of s s' s" occurs in s", such that e'e" can be extended to an s s' s"-element f e' e".
s' ===> s implies e=f, such that e can be extended to an s s"-element in any case.

Trans-u

Let e e" and ef" be s s"-elements. Because of s s" --c--> s' the two elements can be extended to s s' s"-elements e e' e" and e f' f". From s ===> s' follows e'=f' and now the desired equality e"-f" results from s' ===> s".

References

/ 1 / Benecke, K., "An Algebraic Founded Hierarchical Data Model", submitted to ACM TODS

/ 2 / Benecke, K., "The Stroke-List-Operation", Proc. 8. Int. Seminar on Database Management Systems, Piestany 1985

/ 3 / Benecke, K., "Across Data Dependencies to Intelligent Database Operations", appears

/ 4 / Fischer, P.C., Thomas, S.J., "Operators for Non-First-Normal-Form Relations", Proc. IEEE COMPSAL, Nov. 83 pp. 464-475

/ 5 / Jacobs, B.E., "On Database Logic", JACM vol 29, Apr. 82 pp. 310-332

/ 6 / Jacobs, B.E., Walczak, C.A., A Generalized Query-by-Example Data Manipulation Language Based on Database Logic", IEEE Trans. on Software Engineering, Vol. SE-9, No.1, Jan.83, pp. 40-57

/ 7 / Reichel, H., "Structural Induction on Partial Algebras", Akademie Verlag, Berlin 1984

/ 8 / Ullman, J.D., "Principles of Database Systems", Comp. Science Press, Potomac Md., 1980

/ 9 / Shipman, D.W., "The Functional Data Model and the Data Language DAPLEX", ACM TODS, Vol. 6, No 1, March 1981 pp. 140-173

/ 10 / Zlatuska, J., "HIT Data Model a Functional Approach to Databases", Proc. 7. Int. Seminar on Database Management Systems, Varna 1984 p. 30

Proc. IMYCS '86 October 13-17, 1986
Smolenice Castle, ČSSR

APPLICATION OF FERMAT-NUMBER TRANSFORM TO FAST DIGITAL CORRELATION

R. Creutzburg

Academy of Sciences of the G.D.R.
Central Institute of Cybernetics
and Information Processes
P.O.B. 1298
DDR - 1086 Berlin
German Democratic Republic

1. Introduction

With the rapid advances in large scale integration, a growing number of digital signal processing operations becomes attractive. The number-theoretic transform (NTT) was introduced as a generalization of the discrete Fourier-transform (DFT) over residue class rings of integers in order to implement fast cyclic convolution and correlation without round-off errors and with better efficiency than the fast Fourier-transform (FFT) [1,2]. Other interesting applications of the NTT are in fast digital filtering [2], image processing [3], fast coding and decoding of error-correcting codes [4] and very fast FFT computation [5]. A large number of transform methods are developed [1,2,6]. However, it is always a hard problem to find moduli m that are large enough to avoid overflow and to find primitive N-th roots of unity modulo m with minimal binary weight for transform lengths N that are highly factorizable and large enough for practical applications [6]. The Fermat-number transform (FNT) is a compromise between these various conditions [7,8]. In this note the implementation of the Fermat-number transform on a 16-bit computer for 2-dimensional fast digital correlation is described.

2. Number-theoretic transforms

Let Z be the ring of integers and $m > 1$ an odd integer with

prime factorization

$$m = p_1^{r_1} p_2^{r_2} \dots p_s^{r_s}. \qquad (1)$$

Then $a \in Z$ is called primitive N-th root of unity modulo m if [7][10]

$$a^N \equiv 1 \bmod m, \qquad (2)$$

$$\gcd(a^n-1, m) = 1 \quad \text{for every} \quad n = 1,\dots,N-1. \qquad (3)$$

A necessary and sufficient condition for the existence of such primitive N-th roots of unity modulo m is [1]

$$N \mid \gcd(p_1-1,\dots,p_s-1). \qquad (4)$$

Note that (3) is always fulfiled if the modulus m is a prime number.

The NTT of length N with a as primitive N-th root of unity modulo m and its inverse are defined between N-point integer sequences

$$X_n \equiv \sum_{k=0}^{N-1} x_k \, a^{nk} \bmod m, \quad (n = 0,\dots,N-1),$$

$$x_k \equiv N' \sum_{n=0}^{N-1} X_n \, a^{-nk} \bmod m, (k = 0,\dots,N-1),$$

where $NN' \equiv 1 \bmod m$. Note that the components of a signal

$$\underline{x} = (x_0, x_1, \dots, x_{N-1})'$$

have to be quantized to integers before using the NTT. However in many practical applications this is already done by sensors with analog/digital conversion.

The NTT has a similar structure and properties like the DFT, particularly the cyclic convolution property [1].

From the numerical point of view the following three essential conditions on NTT are required [8][9]:

- N has to be large enough and highly factorizable in order to implement fast algorithms like prime-factor-, Winograd-, single-radix-, mixed-radix algorithms [2],
- a should have a simple binary representation (2, for example), so that arithmetic modulo m is easy to perform,
- m has to be large enough to avoid overflow but on the otherhand small enough, so that the machine word length

is not exceeded. Furthermore m should have a simple binary representation.

A list of various NTT's and their parameters a,N,m is given in [6].

3. Fermat-number transform

The Fermat-number transform uses the transform length $N = 2^{d+1}$, the simple binary element a=2 as a primitive N-th root of unity modulo m and the Fermat-number modulus $m = F_d = 2^{2^d} + 1$, $(d \geq 0)$. The one-dimensional Fermat-number transform and its inverse are defined as

$$X_n \equiv \sum_{k=0}^{N-1} x_k \, 2^{nk} \bmod F_d , \quad (n = 0,\ldots,N-1)$$
$$x_k \equiv N' \sum_{n=0}^{N-1} X_n \, 2^{-nk} \bmod F_d , \quad (k = 0,\ldots,N-1) \qquad (5)$$

with $N' \equiv -2^{2^d-d-1} \bmod F_d$.

The transform length N of the Fermat-number transform is always a power of 2, therefore the well-known radix-2 FFT-algorithm can be used [8]. With the help of N additions and N/2 multiplications it is possible to perform the transform (5) of length $N = 2^{d+1}$ as 2 transforms of length N/2. For $N = 2^{d+1}$ this is d-times possible. This gives a number of $N \log_2 N$ additions and $(N/2)\log_2 N$ multiplications for a fast Fermat-number transform. But with the help of a special binary arithmetic for the Fermat-number transform it is possible to avoid any multiplication. Every multiplication in (5) means only a binary shift operation. The special binary arithmetic for the FNT was developed in detail in [8]. The following example shows the application of Fourier- and Fermat-number transforms, respectively, to one-dimensional cyclic convolution.

Example Calculation of cyclic convolution of the signals $\underline{x} = (2,1,0,1)'$ and $\underline{h} = (1,-2,0,0)'$ of length $N = 4$ via discrete Fourier- and Fermat-number transforms.

a) Fermat-number transform: From (5) and with modulus

$F_2 = 2^{2^2} + 1 = 17$ one gets the transform matrix $\underline{T}$ and its inverse $\underline{T}^{-1}$

$$\underline{T} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 4 & 4^2 & 4^3 \\ 1 & 4^2 & 4^4 & 4^6 \\ 1 & 4^3 & 4^6 & 4^9 \end{pmatrix} \equiv \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 4 & -1 & -4 \\ 1 & -1 & 1 & -1 \\ 1 & -4 & -1 & 4 \end{pmatrix} \quad \text{mod } 17,$$

$$\underline{T}^{-1} = 4^{-1} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 4^{-1} & 4^{-2} & 4^{-3} \\ 1 & 4^{-2} & 4^{-4} & 4^{-6} \\ 1 & 4^{-3} & 4^{-6} & 4^{-9} \end{pmatrix} \equiv -4 \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -4 & -1 & 4 \\ 1 & -1 & 1 & -1 \\ 1 & 4 & -1 & -4 \end{pmatrix} \text{ mod } 17,$$

and the Fermat-number transforms of $\underline{x}$ and $\underline{h}$

$$\underline{X} = \underline{T}\,\underline{x} \equiv (4,2,0,2)' \text{ mod } 17,$$

$$\underline{H} = \underline{T}\,\underline{h} \equiv (-1,-7,3,-8)' \text{ mod } 17.$$

Pointwise multiplication

$$\underline{X} \circ \underline{H} \equiv (-4,3,0,1)' \text{ mod } 17$$

and inverse transform gives the cyclic convolution of $\underline{x}$ and $\underline{h}$

$$\underline{y} = \underline{x} * \underline{h} = \underline{T}^{-1}(\underline{X} \circ \underline{H}) \equiv (0,-3,-2,1)' \text{ mod } 17.$$

b) Discrete Fourier-transform : The Fouriermatrix and its inverse are

$$\underline{F} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & j \end{pmatrix}, \quad \underline{F}^{-1} = \frac{1}{4} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & j & -1 & -j \\ 1 & -1 & 1 & -1 \\ 1 & -j & -1 & j \end{pmatrix}.$$

This gives the Fourier-transforms of $\underline{x}$ and $\underline{h}$

$$\underline{X} = \underline{F}\,\underline{x} = (4,2,0,2)'$$

$$\underline{H} = \underline{F}\,\underline{h} = (-1,1+2j,3,1-2j)'.$$

Pointwise multiplication and inverse Fourier-transform gives

$$\underline{X} \circ \underline{H} = (-4,2+4j,0,2-4j)',$$

$$\underline{y} = \underline{x} * \underline{h} = \underline{F}^{-1}(\underline{X} \circ \underline{H}) = (0,-3,-2,1)'.$$

Because of arithmetic in a finite ring one has to pay attention to the overflow modulo m in convolution calculation

$$\underline{y} = \text{INTT}\,(\text{NTT}\,\underline{x} \circ \text{NTT}\,\underline{h}) = \underline{x} * \underline{h}. \qquad (6)$$

From (6) it is necessary that the maximum components $\max|x_n|$ and $\max|h_n|$ of the signals $\underline{x}$ and $\underline{h}$ fulfil

$$|y_n| \leq \max|y_n| \leq N \max|x_n| \max|h_n| \leq (m-1)/2\,, \qquad (7)$$

so that no overflow occurs in (6).

4. Applications

The 2-dimensional Fermat-number transform was implemented on a 16-bit computer in order to perform fast cyclic correlation. A special binary arithmetic modulo the Fermat-numbers $F_4 = 2^{16} + 1$ and $F_5 = 2^{32} + 1$ was developed [8]. With $a = 2$ it is possible to get a transform length $N = 32$ and $N = 64$, respectively. The following table shows the computation times for transform and correlation via Fourier- and Fermat-number transforms for signal-lengths $N = 32$:

Signal	FNT	FFT	Convolution or correlation via	
			FNT	FFT
1-dim.	32 ms	300 ms	100 ms	1 s
2-dim.	2 s	20 s	7 s	70 s

The Fermat-number transform allows the reduction of up to 90 % of computing time!

The method was used to calculate wind velocities from cloud tracking in satellite pictures. The detection of the wind velocity vectors was realized by the determination of the maximum of the crosscorrelation function between a template and the the image field to be searched. The images are produced at a distance of 30 min by a geostationary satellite. Typical small cloud clusters or significant parts of the edges of a cloudy region were used as objects.

5. References

[1] McClellan, J. H., and C. M. Rader: Number Theory in Digital Signal Processing. Prentice Hall, Englewood Cliffs, N. J., 1979

[2] Nussbaumer, H. J.: Fast Fourier Transform and Convolution Algorithms. Springer, Berlin, 1981

[3] Reed, I. S., Truong, T. K., Kwoh, Y. S., and E. L. Hall: Image processing by transforms over a finite field. IEEE Trans. Comput. C-26, 874-881 (1977)

[4] Reed, I. S., Scholtz, R. A., Truong, T. K., and I. R. Welch: The fast decoding of Reed-Solomon codes using Fermat theoretic transforms and continued

fractions. IEEE Trans. Inform. Theory IT-24, 100-106 (1978)

[5] Siu, W. C., and A. G. Constantinides: Very fast discrete Fourier transform, using number theoretic transform. IEE Proceedings G-130, 201-204 (1983)

[6] Creutzburg, R., and M. Tasche: F-Transformation und Faltung in kommutativen Ringen. Elektron. Informationsverarb. Kybernet. 21, 129-149 (1985)

[7] Creutzburg, R., and M. Tasche: Zahlentheoretische Transformationen und primitive Einheitswurzeln in einem Restklassenring modulo m. Rostock. Math. Kolloq. 25, 4-22 (1984), 26, 103-109 (1984)

[8] Creutzburg, R., and H.-J. Grundmann: Schnelle digitale Faltung mittels Fermattransformation. Elektron. Informationsverarb. Kybernet. 21, 35-46 (1985)

[9] Creutzburg, R., and M. Tasche: Number-theoretic tra nsforms of prescribed length. Math. Comp. 47, No.176 (1986),(in print)

[10] Creutzburg, R.: Finite Signalfaltungen und finite Signaltransformationen in endlichen kommutativen Ringen mit Einselement. Dissertation,Wilhelm-Pieck-Universität Rostock, 1984

Proc.IMYCS '86 October 13-17,1986
Smolenice Castle,ČSSR

RUNNING DISCRETE ORTHOGONAL TRANSFORMS

K.O.Egiazarian, S.B.Alaverdian

Computing Centre of Armenian SSR
Academy of Sciences
375044, Sevak Str.1
Yerevan USSR

1. Introduction

This paper is considering a subclass of running discrete transforms - running discrete orthogonal transforms (RDOT) [1] which are containing well known running (or sliding, or moving window, or short-time, or short term [2]-[5]) discrete Fourier transforms. The descriptions of arbitrary quasistationary (slowly changing in time) processes such as speech [4]; action of phase vocoders, spectrographs and some systems of speech recognition [6] are based on the methods of short-time Fourier analysis and synthesis of signals. They are connected with questions such as: filter bank implementation, synthesis of the original sequence by the method of summation of the filter bank outputs or overlap add method, recursive realization of finite impulse filters etc. [1], [3].

By [1] the running discrete orthogonal transform of an arbitrary sequence x(n) with respect to an arbitrary N N orthogonal matrix $H = \|H_{m,k}\|$ is

$$F_m(n) = \sum_{k=0}^{N-1} x(n-k)\, H_{m,k} \; ; \quad m=\overline{0,N-1}; \; n=0,1,... \qquad (1.1)$$

where $H_{m,k}$ is the m,k th element of H.

In general, RDOT can be defined by

$$F_m(n) = \sum_{k=0}^{N-1} x(n-k)\, w(k)\, H_{m,k} \; , \qquad (1.2)$$

where w(n) is a weight function of the "window in the time domain" and can be regarded as the impulse response of a

discrete low-pass filter. RDOT is a function of frequency m and time n.

Existence of a recursive algorithm for computing RDOT of a data sequence in time n from an analogous transform in time n-1, gives a chance for effective calculation of RDOT [1]:

$$F_m(n) = [x(n) - x(n-N)]\, H_{m,0} + \vec{A}_m \vec{F}(n-1) \qquad (1.3)$$

where

$$A = [\vec{A}_0, \vec{A}_1, \ldots, \vec{A}_{N-1}]^T = \frac{1}{N} H U^T H^*, \qquad (1.4)$$

$$\vec{F}(n) = [F_0(n), F_1(n), \ldots, F_{N-1}(n)]^T, \quad m=\overline{0,N-1};\ n=0,1,\ldots;$$

H^* is the conjugate-transposed matrix of H, U^T is the transposition of the circulant matrix U [1].

The matrix A given by (1.4) is the circular advance matrix associated with transform matrix H, and will play a basic role in the sequel.

Let us consider the properties of A-matrices (circular advance matrices) obtained in [1].

Property 1. The effect of interchanging rows i and j of H is to interchange both the i th and j th rows and the i th and j th columns of A.

Property 2. If H is unitary to within a constant k ($H^{-1} = k H^*$), then A is stricly unitary: $A^{-1} = A^*$. Furthermore, the m,n th entry of A^p, say $a^{(p)}_{m,n}$, $0 \le m,n \le N-1$, where p is any integer, p=0, 1, 2,..., is

$$a^{(p)}_{m,n} = k \sum_{l=0}^{N-1} H_{m,l}\, \overline{H}_{n,l-p}\,. \qquad (1.5)$$

Note that due to a large computational complexity of A by (1.5) (of order N^2 operations) it's not advisable to apply for efficient computation of RDOT by (1.3).

Section 2 of this paper derives the analytical expressions of matrices A, corresponding to the well known and most applicable orthogonal bases, defined by matrix H. Section 3 is devoted to efficient algorithms for computing RDOT and estimates of computational complexity. In Section 4 the application of RDOT for recursive realization of FIR (finite impulse response) filters is shown.

2. Analytical expressions of matrices A.

First, we give two properties of matrices A.

Property 3. The multiplication of row j of H by any fixed b is defined by multiplying both the j th row of A by b and j th column of A by $\bar{b}$ (complex conjugate of b).

Property 4. Let the matrices A_1 and A_2 (the A-matrices) correspond to orthogonal matrices H_1 and H_2 of orders m and k. Then the matrix $A = I_m \otimes A_2 - \frac{1}{k}(A_1 - I_m) \otimes V$, where $V = \vec{h}_1 \otimes \vec{h}_k^*$; $H_2 = [\vec{h}_1, \vec{h}_2, \ldots, \vec{h}_k]^T$, will correspond to orthogonal matrix $H = H_1 \otimes H_2$ (the symbol $\otimes$ denotes the Kronecker product [7]).

The Table given below shows the well-known orthogonal bases, which form matrices H and the corresponding matrices A. The analytical expression of matrix A for DEF (see Table) is known [1], the rest ones are considered for the first time (partially by using the property 4). In the Table and further on in the paper e_p is a primitive p th root of unity, say, $e_p = \exp(i\frac{2\pi}{p})$, I_p and J_p are respectively the unit and all-one matrices of order p.

Table

Orthogonal bases (H)	Circular advance matrices A
1) Discrete exponential functions (DEF)	
$H_{1,p} = \|e_p^{jk}\|$; $0 \le j,k \le p-1$	$A_{1,p} = \mathrm{diag}\,(1, e_p^{p-1}, \ldots, e_p)$
2) Functions of Vilenkin-Crestenson (VCF)	$A_{2,p^n} = I_p \otimes A_{2,p^{n-1}} +$
$H_{2,p^n} = H_{1,p}^{(n)} =$	$+ (A_{1,p} - I_p) \otimes V_p^{(n-1)}$,
$= H_{1,p} \otimes \ldots \otimes H_{1,p}$	where $I_1 = 1$; $V_p = J_p A_{1,p}$
3) Functions of Vilenkin-Pontryagin (VPF)	$A_{3,N} = I_{p_k} \otimes A_{3,p_1 \ldots p_{k-1}} +$
$H_{3,N} = H_{1,p_k} \otimes H_{3,p_1 \ldots p_{k-1}}$	$+ \frac{1}{p_1 \ldots p_{k-1}} (A_{3,p_k} - I_{p_k}) \otimes$
$N = \prod_{j=1}^{k} p_j$; $H_{3,p_1} = H_{1,p_1}$	$\otimes V_{p_{k-1}} \otimes \ldots \otimes V_{p_1}$, where $I_1 = 1$,

Continuation of Table

Orthogonal bases (H)	Circular advance matrices A
	$V_{p_j} = J_{p_j} A_{1,p_j}$; $j = \overline{1,k-1}$
4) Functions of Vilenkin-Walsh (VWF) is the VCF when p = 2	$A_{2,2^n} = I_2 \otimes A_{2,2^{n-1}} + \frac{1}{2^{n-2}} T \otimes J_{2^{n-1}} A_{1,2}^{(n-1)}$; where $T = \begin{bmatrix} 0 & 0 \\ 0 & -1 \end{bmatrix}$; $A_{1,2} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$.
5) The cut functions of Vilenkin-Crestenson (CVCF) $H_{5,p^n} = d \begin{bmatrix} 1 & 1 \dots & 1 & \otimes H_{5,p^{n-1}} \\ 1 & e_p \dots & e_p^{p-1} & \otimes I_{p^{n-1}} \\ \vdots & & & \\ 1 & e_p^{p-1} \dots & e_p & \otimes I_{p^{n-1}} \end{bmatrix}$ $H_{5,p} = H_{1,p}$; $d = p^{n-1}$.	$A_{5,p^n} = \mathrm{diag}\,(A_{5,p^{n-1}}, L_{p^{n-1}} + e_p^{p-1} R_{p^{n-1}}, \dots, L_{p^{n-1}} + e_p R_{p^{n-1}})$, where $R = \begin{bmatrix} 0 & 0 \dots 0 & 1 \\ 0 & 0 \dots 0 & 0 \\ \vdots & & \vdots \\ 0 & 0 \dots 0 & 0 \end{bmatrix}$; $L = U^T - R$.

Analytical expressions of circular advance matrices corresponding to other classes of orthogonal bases are obtained as well.

3. The complexity and efficient algorithms of calculating RDOT

We use the analytical expressions of matrices A, found in the previous Section, for the efficient calculation of RDOT by recursive algorithm (1.3).

We estimate the complexity of calculation (1.3), when time interval n is fixed, and A is the circular advance matrix associated with the matrix of an orthogonal basis, say, VCF (see Table). Denote the number of additions and multiplications required for computation (1.3) by s^+ and $s^\times$ respectively. Then $s^+ = N+1+t^+$ and $s^\times = t^\times$, where t^+ and $t^\times$ are the complexities of the computation of matrix expression $A\,\vec{F}(n-1)$. Devide the vector $\vec{F}(n-1)$ into p blocks $\vec{F}(n-1) = [F_0(n-1), \dots, F_{p-1}(n-1)]^T$. According to Table, computation of $A\,\vec{F}(n-1)$ is equivalent to computations

$$\left[A_{2,p^{k-1}} + \frac{1}{p^{k-1}}\left(e_p^{p-j} - 1\right) V_p^{(k-1)}\right] F_j(n-1);\quad 0 \le j \le p-1, \quad (3.1)$$

from where

$$t_k^+ = 2(k-1)p^k - (2k-1)p^{k-1} + p = 2N\log_p \frac{N}{p} - \frac{N}{p}(2\log_p N - 1) + p, \quad (3.2)$$

$$t_k^x = kp^k - (k-1)p^{k-1} - 1 = N\log_p \frac{N}{p} - \frac{N}{p}(\log_p \frac{N}{p} - 1) - 1. \quad (3.3)$$

Note that the computation of $A\vec{F}(n-1)$ where A is the matrix associated with basis VCF and obtained by (1.4), requires $pN \log_p N$ operations of addition and multiplication by using the fast transform algorithm by basis VCF[7]. Hence the calculation of RDOT by (1.3), using the analytical expression of matrix A, is more efficient than the calculation of (1.3) using (1.4).

4. Recursive realization of FIR filters by using RDOT

The recursive realization of FIR filters [8] is the expression of filters output $y(n)$ (which is the convolution of the sequence $x(n)$ of input signals and impulse response $h(n)$) as a linear combination of the running discrete Fourier transforms. Here we consider an efficient algorithm of recursive realization of such filters by using RDOT which has larger economy in computations than the known algorithms [3].

The filtering system of the given impulse response $h(n)$ shown in Fig., where $\vec{F}(n-1) = \left[F_0(n-1), \ldots, F_{N-1}(n-1)\right]^T$, matrix A is defined by (1.4);

$$b_l = \frac{1}{N} \sum_{m=0}^{N-1} h(m)\, \overline{H}_{l,m} \qquad (0 \le l \le N-1) \qquad (4.1)$$

is a discrete orthogonal transform with transform matrix $H = \|H_{m,k}\|_{N-1}$ $(m,k = 0,1,\ldots,N-1)$.

$$y(n) = \sum_{k=0}^{N-1} x(n-k)\, h(k) = \sum_{k=0}^{N-1} x(n-k) \sum_{l=0}^{N-1} b_l\, H_{l,k} =$$

$$= \sum_{l=0}^{N-1} b_l \sum_{k=0}^{N-1} x(n-k)\, H_{l,k} = \sum_{l=0}^{N-1} b_l\, F_l(n) \qquad (4.2)$$

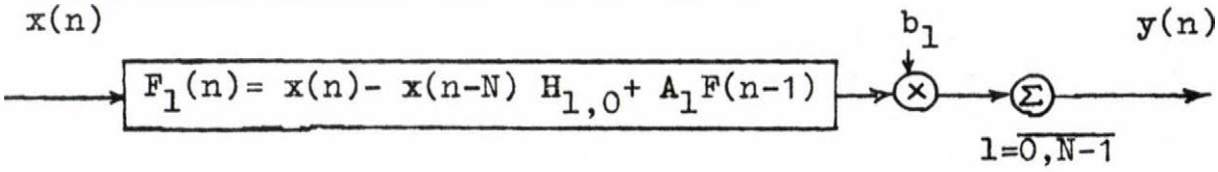


Fig. Filtering system

Note that in the case, when H is a matrix of the basis

DEF, the illustrated filtering system coincides with the known [3]. Thus, let data x(n) be given to the input of system in discrete intervals of time n = 0,1,... The complexity (g^+, g^x) of the realization of this filtering system at each step n, where H is an orthogonal matrix of the basis CVCF (see Table in section 2, expression 5), when p = 2), is $g^+ = 3N + \log N$ and $g^x = 2N + \log N$ instead of $N(\log N + 2)$ and $N(\log N + 2)$ operations of addition and multiplication respectively, in the case of the known filtering system which applies the running discrete Fourier transform [3].

Below we give the program in FORTRAN IV for recursive realization of FIR filters by the described algorithm.

```
      SUBROUTINE  FTCVW(X,H,FO,F1,N,M,Y)
      DIMENSION X(N),H(N),FO(N),F1(N),Y(N)
      NORM = 1
      CALL TCVW(H,N,M,NORM)
      DO 10  I=1,N
      Y(I)=0.
10    FO(I)=0
      DO 11 K=1,N
      XKN=0.
      IF(K.GT.N) XKN=X(K-N)
      DO 12 L=1,N
12    F1(L)=XH(X(K),XKN,L,M)+AMF(FO,L,N,M)
      DO 13 I=1,N
      Y(K)=Y(K)+H(I)*F1(I)
13    FO(I)=F1(I)
11    CONTINUE
      RETURN
      END
```

```
      FUNCTION XH(XK,XKN,L,M)
      XH=XK-XKN
      IF(L.LE.2) GOTO 99
      DO 10 I=1,M-1
      I2=2**I
      IF(L.EQ.I2+1) GOTO 11
10    CONTINUE
      XH=0.
      GOTO 99
11    SQ=FLOAT(I2)
      XH=XH*SQRT(SQ)
99    RETURN
      END
```

```
      FUNCTION AMF(F,L,N,M)
      DIMENSION F(N)
      IF(L.GT.2) GOTO 9
      AMF=F(1)
      IF(L.EQ.2) AMF=-F(2)
      GOTO 99
 9    DO 10 I=1,M-1
      I2=2**I
      IF(L.EQ.I2+1) GOTO 11
10    CONTINUE
      AMF=F(L-1)
      GOTO 99
11    AMF=-F(2*I2)
99    RETURN
      END
```

Acknowledgement

The authors sincerely thank S.S.Agaian for his constructive comments and helpful discussions.

REFERENCES:

1. Staller J.A., Generalized running discrete transforms, IEEE Trans. Acoust., Speech, Signal Processing, vol.ASSP-30, No.1, pp. 60-68, Feb. 1982.
2. Jarmasz M.R., Martens G.O., A simple design for a fast sliding DFT computer, ICASSP 82, Proc. of IEEE Intern. Confer. on ASSP, pp. 502-505, Paris, France,1982.
3. Murakami H., Reed I.S., Recursive realization of finite impulse filters using finite field arithmetic, IEEE Trans. Inform. Theory, vol. IT-23, No.2, pp. 232-242, March 1977.
4. Allen J.B., Short-term spectral analysis and synthesis and modification by discrete Fourier transform, IEEE Trans. Acoust., Speech, Signal Processing, vol.ASSP-25, No.3, pp. 235-238, June 1977.
5. Allen J.B., Rabiner L.R., A unified approach to short-time Fourier analysis and synthesis, Proc. IEEE, vol.65, No.11, pp. 1558-1564, Nov. 1977.
6. Schafer R.W., Rabiner L.R., Digital representation of speech signals, Proc. IEEE, vol.63, No.4, pp. 662-677, April 1975.
7. Dagman E.E., Kuharev G.A., Fast discrete orthogonal transforms, Novosibirsk: Nauka, 1983 (on russian).
8. Rabiner L.R., Gold B., Theory and application of digital signal processing, Prentice-Hall, Inc., New Jersey, 1975.

Proc. IMYCS '86 October 13-17, 1986
Smolenice Castle, ČSSR

RELATIONS BETWEEN ATTRIBUTE GRAMMARS AND HORN CLAUSES

P. Forbrig

Sektion Informatik
Wilhelm-Pieck-Universität Rostock
2500 Rostock
DDR

1. Introduction

Logic programming and attribute grammars have been studied since the end of the 1960s. Fundamental papers were presented by Kowalski [7] and Knuth [4].
The computational formalisms have been developed independently with different motivations. But results of the 1980s show, that attribute grammars and logic programs are closely related. The aim of this paper is to discuss relationships between both formalisms and to demonstrate their application in data driven software design.
To illustrate some principles of these formalisms we will use a simple example, a telegram problem.
A program is required to process a stream of telegrams. This stream is available as a sequence of words, spaces and the special delimiter *, showing the end of a telegram. The stream is terminated by the occurrence of the empty telegram.
The telegrams are to be processed to determine for each telegram the number of words and the number of long words with more than twelve characters. The telegrams together with the statistics have to be stored on an output file by eliminating all but one spaces between words. The longest possible word has twenty characters. For simplicity telegram streams containing words with more than twenty characters are omitted.
In the following more formal specifications the spaces will be presented by #.

2. Logical Programming

For the definition of the telegram problem a finite system of Horn clauses is used (see e. g. Loyd [9]).
A Horn clause is a string of the form

$$B \leftarrow A_1, \ldots, A_m \quad (m \geq 0) \qquad (+)$$

where B, A_1, ... , A_m are atoms.
An atom consists of an n-ary predicate symbol followed by a list of n terms inserted in paranthesis.
Let x_1, ... , x_k be the only variables occurring in the terms of B, A_1, ... , A_m. Then (+) means

$$(\forall x_1, \ldots, x_k)\ (A_1 \ \ldots \ A_m \Longrightarrow B).$$

By usual interpretation of formulas we get:

> For all values of the variables x_1, ... , x_k such that all A_i are valid B is valid too. (A_i and B are assertions arising from A_i and B respectively.)

For the definition of facts a special kind of Horn clauses is used:

$$B \leftarrow .$$

To achieve better readability of the clauses specifying our telegram problem we will first give an interpretation of the atoms.

sum(X,Y,Z)	- Z is the sum of X and Y.
less(X,Y)	- X is less than Y.
character(X)	- X is a character.
word(X,L)	- X is a word consisting of L characters.
telegram(X,Y)	- Y is the output telegram corresponding
tel(X,Y)	to the input telegram X.
telegramstream(X,Y)	- Y is the output telegram stream corres-
telstr(X,Y)	ponding to the input telegram stream X.

The atoms tel and telstr are only used to garantee that a telegram stream cannot consist of the empty telegram.
For simplicity we omit the definition of clauses referring to natural numbers and the concatenation of terms. They are supposed to be predefined.

Under this precondition the following Horn clauses specify the telegram problem:

```
character( X )  ⟵ . X ∈ {A,..., Z, a,..., z, 0,..., 9}
word( X,1 )     ⟵ character( X ).
  {A word consisting of one character only has the length 1.}
word( XY,L1)  ⟵ character( X ), word( Y,L ), sum( L,1,L1 ).
  {A word consisting of one character and a word of length L
   has the length L + 1.}
telegram( #X,Y )  ⟵ telegram( X,Y ).
  {If Y is the output telegram of X then Y is also the output
   telegram of =X.}
telegram( X#Y,X#Z#N1#L# )  ⟵ word( X,L0 ), less( L0,13 ),
    tel( Y,Z#N#L# ), sum( N,1,N1 ).
  {A telegram consisting of a short word X and a telegram Y has
   the output telegram X#Z with N+1 words and L long words, if
   Z is the output telegram of Y and Z has N words and L long
   words.}
telegram( X#Y,X#Z#N1#L1# )  ⟵ word( X,L0 ), less( 12,L0 ),
    less(L0,21), tel( Y,Z#N#L# ), sum( N,1,N1 ), sum( L,1,L1 ).
  {A telegram consisting of a long word X and a telegram Y has
   the output telegram X#Z with N+1 words and L+1 long words,
   if Z is the output telegram of Y and Z has N words and L
   long words.}
tel( *,*#0#0# )  ⟵ .
tel( X,Y )       ⟵ telegram( X,Y ).
tel( #X,Y )      ⟵ tel( X,Y ).
telstr( *,*#0#0# )  ⟵ .
telstr( XI,Y0 )     ⟵ telegram( X,Y ), telstr( I,0 ).
telegramstream( XI,Y0 )  ⟵ telegram( X,Y ), telstr( I,0 ).
```

For a given input telegram stream T the corresponding output telegram stream is determined (if it exists) beginning from the goal telegramstream(T,Y).
This is done by constructing a proof for the goal by Horn clauses. The variables of the Horn clauses are suitably substituted and the determined value of the variable Y of the goal is the desired output telegram stream.

3. Specification by attribute grammars

We use a generative version of attribute grammars, the extended attribute grammars introduced by Watt and Madsen. The interested reader is referred to [16]. We will only give a short

introduction into the formalism. An extended attribute grammar is a 5-tuple $G = (D, V, Z, B, R)$ and its elements are defined as follows.

- $D = (D_1, D_2, \ldots , f_1, f_2, \ldots)$ is an algebraic structure with domains D_1, D_2,... and partial functions f_1, f_2,... operating on Cartesian products of these domains. Each object in one of these domains is called attribute.
- V is the vocabulary of G, a finite set of symbols partitioned into nonterminals V_n and terminals V_t. Associated with each symbol in V is a fixed number of attribute positions, classified as either inherited ($\downarrow$) or sythesised ($\uparrow$).
  An attribute expression is
  a) a constant attribute, or b) an attribute variable, or
  c) a function application $f(e_1,\ldots,e_n)$, where $e_1,\ldots,e_n$ are attribute expressions and f is chosen from D.
  Attribute expressions can be written on attribute positions.
- Z is the start symbol, a member of V_n.
- B is a finite collection of attribute variables.
- R is a finite set of production rule forms $F ::= F_1 \ldots F_k$, where $F_1, \ldots , F_k$ are attribute symbols from V and F is an attributed symbol from V_n. Provided all attribute expressions have defined values we get a production rule $A::=A_1 \ldots A_k$.

To specify the telegram problem by an extended attribute grammar it is possible to use following domains and functions. The functions define the concatenation of strings of different type. The attribute variables at the beginning of each line have the corresponding domains.

```
ITS: ITELSTREAM = {* | cs( ITEL, ITELSTREAM)}
       {Input telegram stream}
OTS: OTELSTREAM = {cat(+,#,0,#,0,#) | cos( OTEL, OTELSTREAM)}
       {Output telegram stream}
IT:  ITEL = {* | ct( WORD,#,ITEL ) | ce( #, ITEL )}
       {Input telegram}
OT:  OTEL = {cot( AOTEL,#,INT,#,INT,# )}
       {Output telegram}
     AOTEL = {* | ca( WORD,#,AOTEL )}
       {Auxiliary output telegram}
```

W: WORD = {CHARACTER | c(CHARACTER, WORD)}

C: CHARACTER = {A|...|Z|a|...|z|0|...|9}

N, N1, L, L1, LONG: INT = {0 | 1 | 2 ... }

The telegram problem is specified by the following rules:

<character ↓X> ::= . X c A,..., Z, a,..., z, 0,..., 9

<word ↓W ↑1> ::= <character W .
<word ↓c(C,W) ↑L1> ::= <character ↓C> <word ↓W ↑L>
<sum ↓L ↓1 ↑L1> .

<telegram ↓ce(#,IT) ↑OT> ::= <telegram ↓IT ↑OT>.
<telegram ↓ct(W,#,IT) ↑cot(ca(W,#,OT),#,N1,#,L,#)> ::=
<word ↓W ↑LONG><less ↓LONG ↓13>
<tel ↓IT ↑cot(OT,#,N,#,L,#)><sum ↓N ↓1 ↑N1>.
<telegram ↓ct(W,#,IT) ↑cot(ca(W,#,OT),#,N1,#,L1,#)>::=
<word ↓W ↑LONG><less ↓12 ↓LONG><less ↓LONG ↓21>
<tel ↓IT ↑cot(OT,#,N,#,L,#)><sum ↓N ↓1 ↑N1>
<sum ↓L ↓1 ↑L1>.

<tel ↓* ↑cot(*,#,0,#,0,#)> ::= .
<tel ↓IT ↑OT> ::= <telegram ↓IT ↑OT>.
<tel ↓ce(#,IT) ↑OT> ::= <telegram ↓IT ↑OT>.

<telstr ↓* ↑cot(*,#,0,#,0,#)> ::= .
<telstr ↓cs(IT,ITS) ↑cos(OT,OTS)> ::=
<telegram ↓IT ↑OT><telstr ↓ITS ↑OTS>.

<telstream ↓cs(IT,ITS) ↑cos(OT,OTS)> ::=
<telegram ↓IT ↑OT><telstr ↓ITS ↑OTS>.

The nonterminals sum and less are supposed to be predefined. This specification is very similar to the set of Horn clauses and no comment is necessary.

We have used both formalisms in the following data driven manner:

1. The structure of input data was described by grammar rules (Horn clauses). In this first step relations between attribute positions (terms) were neglected.
2. Nonterminals and functions (atoms) were introduced to describe relations between attribute positions (terms). Sometimes it is necessary to copy rules (clauses), if different semantical relations exist. (see e.g. counting normal and long words)
3. The newly introduced nonterminals (atoms) were defined by rules (clauses). (This is not necessary if nonterminals (atoms) are predefined.)

The similarity of these methods and the resulting specifications raises the question, whether these both formalisms are very closely related.

4. Relations between logical programming and grammars

Recently some very interesting results about attribute grammars and logical programming have been published. Figure 1 is an attempt to give an overwiew of some of these papers.

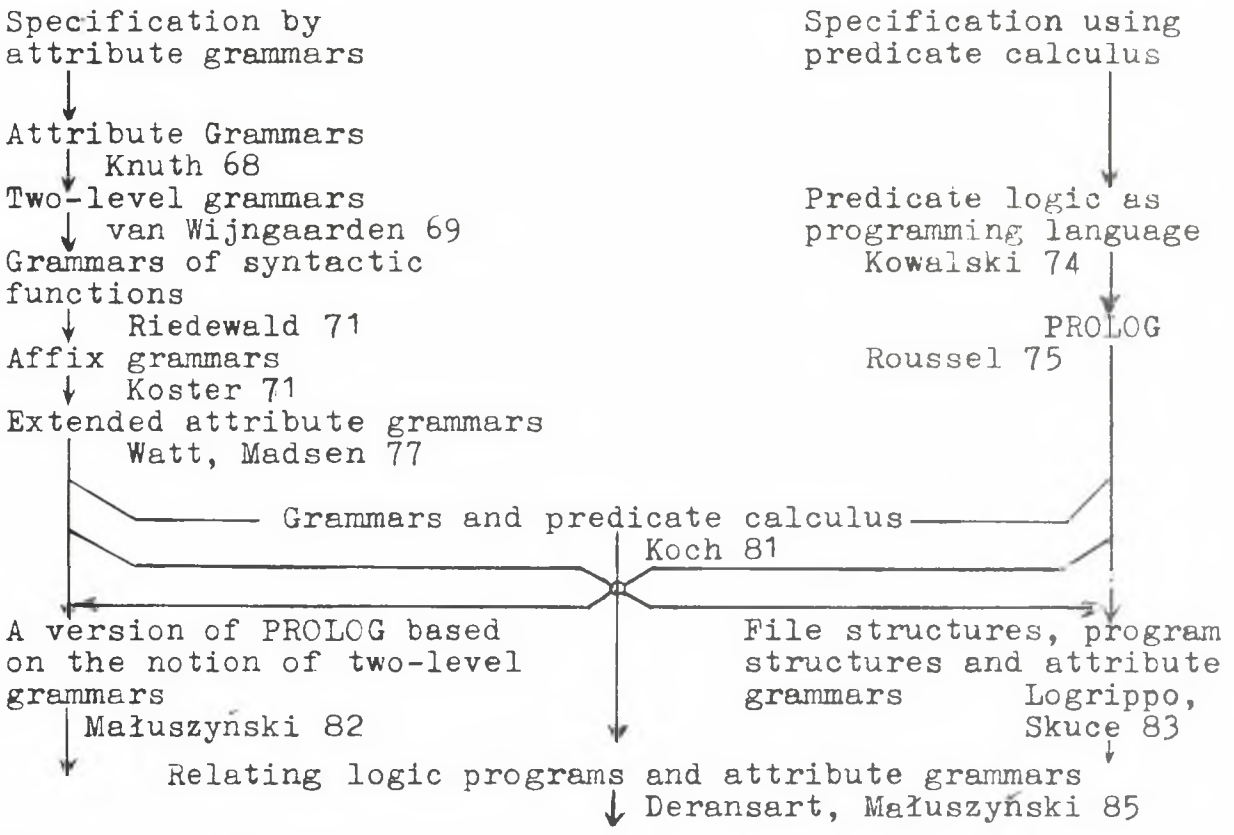


Figure 1 Papers concerning attribute grammars and logical programming

As Figure 1 shows there exist relations between attribute grammars and logical programs, which were developed independently with differen motivations. Attribute grammars were introduced in the 1960s to define programming languages and their compilers. At the same time the predicate logic was applied to programming. The first investigations were summarized in the paper by Kowalski and resulted in the programming language PROLOG. Relationships between both formalisms were studied by Koch [5] and it was shown that both concepts are able to compute relations. Małuszyński ([10] and [11]) has proved, that a special class of two-level grammars is able to implement PROLOG in an efficient way. He introduced the concept of grammatical unification, where terms are described by contextfree grammars.

Logrippo and Skuce [8] follow a contrary way and implement an attribute grammar using PROLOG. Deransart and Małuszyński [1] show that a set of Horn clauses can be transformed into a semantical equivalent functional attribute grammar, and vice versa. Semantically equivalent means, that both specifications compute the same relation. Deransart and Małuszyński use a slightly different definition of attribute grammars. They do not define the direction of attribute propagation within the production rules, but use a separate direction assignment for nonterminals. It was proved that semantics of attribute grammars does not depend on these direction assignments.
Nevertheless, the direction assigned to positions in attribute grammars provide a pragmatic information which makes it possible to organize the attribute evaluation more efficiently. This principle can also be used in connection with Horn clauses. The results of these investigations show that it is possible to use methods of one of the formalisms for solving problems related to the other. So it was possible

- to show that proof trees of logic programs and the decorated syntax trees of attribute grammars have a similar structure.
- to find a sufficient condition under which no infinite term can be created during the computation of a logic program.
- to define a nontrivial class of logic programs which can run without employing unification in its general form.

However, in contrast to a set of Horn clauses an attribute grammar defines not only a relation but also a language. Definite clause grammars [12] are an extension of Horn clauses. Like attribute grammars, definite clause grammars define a language. In Małuszyński's approach the language defined by an attribute grammar is used to control the computation of the relation specified by the grammar. In cooperation with Riedewald we have used in [15] a similar concept to specify a telegram problem. The language defined by the grammar is the set of all correct input telegram streams and during attribute propagation the corresponding output telegram stream is computed.
We have compared this approach with algebraic and denotational specifications.

The formalism of attribute grammars was extended by abstract data types. This allows the combination of data driven program development and data abstraction, usually classified as contrary.

5. Conclusion

Attribute grammars and Horn clauses are closely related formalisms to compute relations. It is possible to use methods of the formalisms for solving problems to the other. Some of these results were discussed and the data driven application of both formalisms was demonstrated.
We believe that attribute grammars should be further investigated for theory and application in software design.

References

[1] Deransart, P.; Małuszyński, J.: Relating Logic Programs and Attribute Grammars, INRIA, Report, 1985.
[2] Forbrig, P.: Kombination der datengesteuerten Programmierung nach Jackson mit der Datenabstraktion nach Parnas, WPU Rostock, Sektion Informatik, Report, 1985.
[3] Forbrig, P.: Datengesteuerte logische Programmierung und attributierte Grammatiken, Rost. Inform. Heft 4, 1986.
[4] Knuth, D. E.: Semantics of Context-free Languages, Mathem. Systems Theory, 2(2)1968, P. 127-145.
[5] Koch, G.: Grammars and Predicate Calculus, Univ. Kopenhagen, Report DIKO 81/16, 1981.
[6] Koster, C. H. A.: Affix Grammars, ALGOL68 Implementation, (Ed. Peck), North Holland, 1971, P. 95-109.
[7] Kowalski, R. A.: Predicate Logic as Programming Language, Information Processing, 1974, P. 569-574.
[8] Logrippo, L.; Skuce, D. R.: File Structures, Program Structures and Attribute Grammars, IEEE Trans. on Software Engineering, 9(3)1983, P. 260-267
[9] Loyd, J. W.: Foundations of Logic Programming, Springer Verlag, Heidelberg - New York - Tokio, 1984.
[10] Małuszyński, J.: A Version of PROLOG Based on the Notion of Two-level Grammars, Linkoeping, 1982.
[11] Maluszynski, J.: Towards a Programming Language Based on the Notion of two-level Grammars, Theoretical Computer Science, 28(1/2)1984, P. 13-45.
[12] Pereira, F.C.N.; Warren, D.H.D.: Definite Clause Grammars For Language Analysis, Artif. Intelligence 12, 1980.
[13] Riedewald, G.: Internal Report, WPU Rostock, 1971.
[14] Riedewald, G.; Małuszyński, J.; Dembiński, P.: Formale Beschreibung von Programmiersprachen, Berlin, 1983.
[15] Riedewald, G.; Forbrig, P.: Software Specification Methods and Attribute Grammars, submitted to Acta Cybernetica.
[16] Watt, D. A.; Madsen, O. L.: Extended Attribute Grammars, University of Glasgow, Report 10, 1977

Proc. IMYCS '86 October 13-17, 1986
Smolenice Castle, ČSSR

FIXED POINT LANGUAGES OF RATIONAL TRANSDUCTIONS

Wit Forys

Institute of Computer Science
Jagiellonian University
31-501 Kraków
Poland

1. Introduction

This paper concerns the study of fixed points of rational transductions rational relations . A set of fixed points of a rational transduction as a subset of a free monoid is called a fixed point language. Our paper is closely related to the results on fixed point and equality languages and is a sequel to K. Culik II [3], K. Culik II, J. Karhumäki [4], J. Engelfriet, G. Rozenberg [6], [7]. The results may be treated as a contribution to the qualitative theory of equations in free monoids. It seems to us that a motivation for those investigations is quite clear from the mathematical point of view and also within the formal language theory. If not, see J. Engelfriet, G. Rozenberg [6], [7].

2. Preliminaries

We assume the familiarity with the basic notions of formal languages and semigroup theories. For any undefined notions see S. Eilenberg [5], G. Lallement [8]. Now we recall only some basic for our paper definitions and notations. Let M be any monoid. The family $\mathrm{Rat}\, M$ of rational subsets of M is the least family R of subsets of M satisfying the following conditions:

1^{o} $\emptyset \in R$, $\{m\} \in R$ for any $m \in M$

2^{o} if $A, B \in R$ then $A \cup B$, $A \cdot B \in R$

3^0 if $A \in R$ then $A^+ = \bigcup_{n=1}^{\infty} A^n \in R$

Now let A be any finite non empty set (an alphabet). A^*/A^+ denotes a free monoid/semigroup generated by A. For any word $w \in A^*$ the length of it is denoted by $|w|$. Let $a \in A$ and $w \in A^*$. By $\#_a w$ is denoted the number of occurrences of the letter a in the word w. A subset $L \subset A^*$ is called regular iff there exist a finite monoid M and a morphism $\varphi : A^* \longrightarrow M$ such that $L = \varphi^{-1}(\varphi(L))$. For any finite generated free monoid A^* the family of all regular subsets of A^* coincides to the family Rat A^* (Kleene theorem). For any set S its cardinality is denoted by $\#S$. Let $h : A^* \longrightarrow B^*$ be any morphism where A^*, B^* free monoids. A morphism h is said:

- alphabetic if $h(A) \subset B \cup \{1\}$
- strictly alphabetic if $h(A) \subset B$.

A rational subset t of a monoid $A^* \times B^*$ where A and B are any alphabets is called a rational relation. Such relation can be treated as a function from A^* into the set $\mathcal{P}(B^*)$ putting $t(w) = \emptyset$ if $(w, B^*) \cap t = \emptyset$. In this case t is called a rational transduction and usually is written as $t : A^* \longrightarrow B^*$. The following characterization of rational transductions is due to M. Nivat:

THEOREM 2.1. The following conditions are equivalent

1^0 $t : A^* \longrightarrow B^*$ is a rational transduction

2^0 There exist an alphabet Z, two alphabetic morphisms $h : Z^* \longrightarrow A^*$, $g : Z^* \longrightarrow B^*$ and a regular language $R \subset Z^*$ such that

$$t(w) = g(R \cap h^{-1}(w)) \quad \text{for any } w \in A^*.$$

It is possible to assume that R is a local regular language. Hence any rational transduction t can be factorized as follows

$$t = g \circ \cap R \circ h^{-1}$$

where $\cap R : \mathcal{P}(Z^*) \longrightarrow \mathcal{P}(Z^*)$ is an operation defined by $\cap R(X) = R \cap X$ for any $X \in \mathcal{P}(Z^*)$.

A rational transduction t is said:

- increasing if for any $v \in t(w)$ $|w| \leq |v|$ holds
- decreasing if for any $v \in t(w)$ $|w| \geq |v|$ holds

- length preserving if for any $v \in t(w)$ $|w| = |v|$ holds
- continuous if for any $w \in A^+$ $t(w) \subset A^+$ holds.

Let $t:A^* \longrightarrow A^*$ be any rational transduction. A set of all fixed points of t defined as follows

$$Fp\ t = \{ w \in A^* : w \in t(w) \}$$

is equal to

$$Fp\ t = h\left(Eq(h,g) \cap R\right)$$

where $Eq(h,g)$ is an equality set of morphisms h and g. We assume that for any rational transduction $t = g \circ \cap R \circ h^{-1}$ there is no $a \in A$ such that $g(a) = h(a) = 1$ (if such "a" exists it can be deleted).

3. General Case

Now let us recall that a morphism $h:A^* \longrightarrow B^*$ is linear bounded on $L \subset A^*$ if there exists an integer $k \geqslant 0$ such that $|h(w)| \leq k|w|$ for each $w \in L$. It is true that the family of context-sensitive languages is closed under the linear bounded morphisms A. Salomaa [10]. Now we have the following

THEOREM 3.1. Let $t:A^* \longrightarrow A^*$ be any rational transduction. A set $Fp\ t$ is a context-sensitive subset of A^*.

PROOF. As was stated above $Fp\ t = h(R \cap Eq(h,g))$ where g,h are alphabetic morphisms. A set $Eq(h,g)$ is a context-sensitive subset of Z^* J. Engelfriet, G. Rozenberg [7]. Let $\{a \in A : h(a) = 1\} = \{c_1,\ldots,c_k\}$. For each c_i, $i = 1,\ldots,k$ we define a morphism $h_{c_i}:A^* \longrightarrow A^*$ putting

$$h_{c_i}(a) = \begin{cases} 1 & \text{for } a = c_i \\ a & \text{otherwise} \end{cases}$$

Any h_{c_i} is a linear bounded morphism on the set $L = Eq(h,g) \cap R$. So $h_{c_i}(L)$ is a context-sensitive set and consequently

$$K = h_{c_k} \circ h_{c_{k-1}} \circ \ldots \circ h_{c_1}(L)$$

is a context-sensitive subset of A^*. Let $\bar{A} = A \setminus \{c_1,\ldots,c_k\}$. Of course $K \subset \bar{A}^*$. Now we define a morphism $\bar{h}:\bar{A}^* \longrightarrow A^*$ by the condition $\bar{h}|_{\bar{A}} = h|_{\bar{A}}$. The following equalities are true:

$$\bar{h}(K) = h(L) = Fp\ t.$$

But $\bar{h}$ as a strictly alphabetic morphism is linear bounded

on K. Hence finally we obtain that $\bar{h}(K) = Fp\ t$ is a context-sensitive subset of A^*.

REMARK 3.2. Let $t:A^* \longrightarrow A^*$ be any rational transduction such that $t = g \circ \cap R \circ h^{-1}$ where $h:Z^* \longrightarrow A^*$, $g:Z^* \longrightarrow A^*$ are alphabetic morphisms and $\#Z = 2$. In this case $Fp\ t$ is a regular subset of A^*.

We justify the above remark as follows. A set $Eq(h,g)$ for morphisms h,g over a binary alphabet $Z = \{a,b\}$ is regular or is of the form K. Culik II, J. Karhumäki [4] :

$$\{1\} \cup \left\{ w \in Z^* : \frac{\#_a w}{\#_b w} = k \right\}$$

where $k > 0$, $k \in Q$. In view of our assumption h and g are alphabetic morphisms. So $Eq(h,g)$ is a regular set or is of the form $\{1\} \cup \{ w \in Z^* : \#_a w = \#_b w \}$. If $Eq(h,g)$ is regular the statement is trivially true. So we consider the second case. It takes place iff h and g are of the form (within isomorphisms): $h(a) = 1$, $h(b) = b$, $g(a) = b$, $g(b) = 1$. In that case the set $Eq(h,g)$ is the Dyck language over $Z = \{a,b\}$. So it is a context-free subset of Z^*. Then $Eq(h,g) \cap R$ is context-free also. But $h(Eq(h,g) \cap R)$ is context-free over one elementary alphabet and thus regular.

The argumentation for the following remark may be computer aided. For this purpose we have used SCHNEIDER microcomputer CPC 64 k.

REMARK 3.3. Let $t:A^* \longrightarrow A^*$ be any rational transduction such that $t = g \circ \cap R \circ h^{-1}$ where $h:Z^* \longrightarrow A^*$, $g:Z^* \longrightarrow A^*$ are alphabetic morphisms and $\#Z = 3$. In this case $Fp\ t$ is a context-free subset of A^*.

The justification is as follows. For alphabetic morphisms h and g over the alphabet $Z = \{a, b, c\}$ it is possible to determine all possible forms of the set $Eq(h,g)$. With the computer aid we have analysed this problem and have obtained the following results, beyond the cases in which $Eq(h,g)$ is regular. The set $Eq(h,g)$ is equal to (within isomorphism):

$1^o\ L_1^*$ where $L_1 = \{w \in Z^* : \#_b w = \#_c w\}$

$2^o\ L_2^*$ where $L_2 = \{w \in Z^* : \#_a w = \#_b w + \#_c w\}$

$3^o\ (K_1 \cup K_2)^*$ where $K_1 = \{a^n b^n c^n \in Z^* : n = 1,2,\ldots\}$

and $K_2 = \{c^n b^n a^n \in Z^* : n = 1,2,\dots\}$

First we consider the cases 1^o and 2^o together. Let D be the Dyck language over a binary alphabet $\{x,y\}$. Let for $i = 1,2$ $f_i : Z^* \longrightarrow \{x,y\}^*$ be morphisms defined as follows: $f_1(a) = 1$, $f_1(b) = x$, $f_1(c) = y$ and $f_2(a) = x$, $f_2(b) = f_2(c) = y$. Then L_1 and L_2 are inverse images of D under the morphisms f_i respectively. Hence are context-free. Consequently in the cases 1^o and 2^o the set Fp t is context-free.

The case 3^o follows from the fact that in the factorization of t it is possible to use a local regular set $R \subset Z^*$ and that this case holds for a morphism h which ereases a letter "b". Basing on this fact one can consider all possible forbidden and authorized transitions in R and check on $(K_1 \cup K_2)^* \cap R$ for all these possibilities. The number of them can be reduced by the facts that some are symmetric and other lead to the empty set. Finally, after the analysis of all subcases we have came to the conclusion that the set $(K_1 \cup K_2)^* \cap R$ is regular (mostly) or it is obtained from K_1 and K_2 by the operations: $\cup$, $\cdot$, *. For example: $K_1 L_3^* K_2$, K_1^*, $K_1 L_3^*$, $(K_1 K_2)^* \cdot (K_1 \cup 1) \cup (K_2 K_1)^* \cdot (K_2 \cup 1)$ where $L_3 = K_1 \cup K_2$. Of course $h(K_1)$ and $h(K_2)$ are context-free. The class of context-free languages is closed under the above operations. Hence $h((K_1 \cup K_2)^* \cap R)$ is context-free because of the mentioned properties and that $h(X \cup Y) = h(X) \cup h(Y)$, $h(X^*) = h(X)^*$, $h(XY) = h(X)h(Y)$.

The following example answers the question for Z of the cardinality 4.

EXAMPLE. Let $Z = \{a,b,c,d\}$, $R = Z^*$ and
$h(a) = 1$, $h(b) = a$, $h(c) = b$, $h(d) = c$
$g(a) = a$, $g(b) = b$, $g(c) = c$, $g(d) = 1$.
For $t = g \circ \cap R \circ h^{-1}$ it holds

$$\text{Fp } t = \{a^n b^n c^n,\ c^n b^n a^n : n = 1,2,\dots\}^*$$

and this set is context- sensitive indeed.

4. Some Special Cases

We end our considerations by listing some types of rational transductions which have a regular set of fixed points.

Let t be any rational transduction. If t is increasing or decreasing or length preserving or continuous then Fp t is regular. For any rational transduction of the above type there exists a factorization with at least one strictly alphabetic morphism L. Boason, M. Nivat [2], J. Leguy [9]. This implies the regularity of Eq(h,g) in view that another morphism is alphabetic and thus proves the statement.

References

[1] J. Berstel, Transductions and context-free languages, Teubner Verlag, 1979
[2] L. Boason, M. Nivat, Sur diverses familles de langages fermées par transduction rationnelle, Acta Informatica 2, 1973
[3] K. Culik II, Homomorphisms: decidability, equality and test sets in Formal Language Theory, R.V. Book ed. 1980
[4] K. Culik II, J. Karhumäki, On the equality sets for homomorphisms on free monoids with two generators, RAIRO 14, 1980
[5] S. Eilenberg, Automata, languages, and machines vol. A, Academic Press 1974
[6] J. Engelfriet, G. Rozenberg, Equality languages and fixed point languages, Inf. Control 43, 1979
[7] J. Engelfriet, G. Rozenberg, Fixed point languages, equality languages and representation of recursively enumerable languages, J. Assoc. Comp. Mach. 27, 1980
[8] G. Lallement, Semigroups and combinatorial applications, Wiley 1979
[9] J. Leguy, Transductions rationnelles décroissantes, RAIRO 15, 1981
[10] A. Salomaa, Formal languages, Academic Press 1973

Proc.IMYCS '86 October 13-17,1986
Smolenice Castle,ČSSR

AN ALGEBRAIC VIEW ON THE CLASSES OF BINARY IMAGES

M.Ftáčnik

Department of Theoretical Cybernetics
Comenius University
842 15 Bratislava
Czechoslovakia

Digital image processing or picture processing is a rapidly growing discipline with broad applications. Most of the work involves picture analysis (given a picture to construct its desciption) or simply picture classification (to determine a class to which a given picture belongs), both together called also picture recognition. The picture classification utilizes the so called features of the digital picture. The aim of this contribution is to show relationship between the special kind of the features of binary images and some algebraic properties of these images.

1. A picture is input to the computer by sampling its brightness values at discrete grid of points and digitizing or quantizing these values to a finite number of binary places. The result of this process is called a digital picture (image); it is a rectangular array of discrete values. The elements of this array are called pixels and the value of a pixel is called its level. If the level of each pixel can take only two values the digital picture is called binary picture (image). Usually is one value denoted by unit (representing the pixels of an object) and the second by zero (representing the backround). In fact in binary images the silhouettes of the objects are available and the experience shows that the information important from the point of recognition is concentrated on the border of the silhouette. When reco-

gnizing these objects various morphometric features are utilized such as the surface, diameter, height or width of an object. As the very interesting for recognition from the psychological point of view are the points corresponding to the greatest curvature of the silhouette´s border, particulary the points in which the direction of the border is changing. The convenience of these points for the description of the object´s form was investigated by Duda and Hart in [1] and for the evaluation of the combinational complexity of the matrix by Haverlík in [4].

These points are called corner points. Before we introduce them formally we have to define the plane equivalent of the digital picture. The similar definition was presented in [2].

Definition 1. Let A be a binary picture containing $n \times n$ pixels and let d be a constant equal to 1/2. The p-grid of picture A is called a finite square grid in the plane where the centre of the square with coordinates (i,j) is corresponding to the pixel with coordinates (i,j) and the square is determined by the grid points with coordinates $(i \pm d, j \pm d)$. If the pixel´s level is equal to 1 then the corresponding square is denoted as black square.

Then the corner point can be defined as follows.

Definition 2. The point of the p-grid of binary picture A is called the corner point if the number of black squares determined by this point is odd.

2. There is a great interest in discovering the basic properties of binary images which can be utilized in image processing itself. The basic topological properties were studied, involving such concepts as adjacency and connectedness. The geometrical properties studied depend especially on the positions of the pixels of an object. Last but not least are the algebraic properties of binary images.

In what follows we suppose the binary image is represented by a Boolean matrix the elements of which will be farther denoted as pixels and a_{ij} will denote the level of the pixel with coordinates (i,j). There is no problem to show that

the class of all square Boolean matrices of order N (denoted as $\mathcal{M}_N$) with operation $\oplus$ (sum modulo 2) forms with the field $T = \{0,1\}$ with operations $\oplus$ and the logical product the vector space. The canonical base of this space is the set of matrices of type $A = (a_{ij})$, where $a_{ij} = 1$ if $i = r$ and $j = s$ exactly for one pair $r,s \in \{1, \ldots, N\}$ and $a_{ij} = 0$ otherwise. The cardinality of this base is N^2. For the vector space $(\mathcal{M}_N, \oplus)$ also other bases can be found fulfilling some restrictions, e.g. one can require every element of the base to be monotone.

In [2] we studied the class of monotone matrices which can be defined as follows.

Definition 3. The matrix $A \in \mathcal{M}_N$ is called monotone (i.e. it belongs to the class of monotone matrices E_N) if for its p-grid at least one of the following conditions is satisfied ($d = 1/2$):

1) in p-grid of A such a 4-connected path ξ (see e.g. [5]) can be chosen which consists of 2N+1 points of the grid and connects the point (d,d) with the point (N+d,N+d) and black are only the squares (i.e. in matrix A the value 1 have only the pixels) with coordinates (i,j) for which $j < q(i)$ and $q(i) = \max\{y \mid (i - d, y) \in \xi\}$;

2) the condition 1 is fulfilled for a matrix B which can be obtained by rotating the matrix A at $\pi/2$, π or $3\pi/2$ in clockwise orientation (Fig. 1).

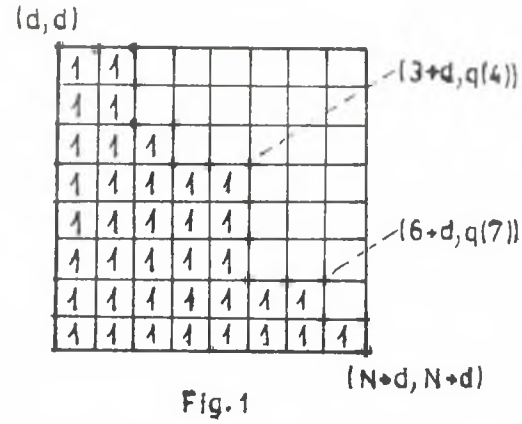


Fig.1

The monotone matrices represent such binary pictures which contain the part of an object with the "simple" border

(in fact it is an approximation of the simple border). This border is determined by the path from definition 3.

From the results presented in [3] it follows that E_N is a generating set for the class $\mathfrak{N}_N$. From the well-known theorem for the finite dimensional spaces it follows that from a generating set of the space the base of that space can be chosen.

So the requirement of the monotonicity of every base element is consistent and we shall present such a base. The most simple one can be defined as follows. To every element A of the canonical base CB in which exactly one $a_{ij} = 1$ there is assigned a matrix $B = (b_{pq})$ such that $b_{pq} = 1$ if $p \geqslant i$ and $q \geqslant j$. It is obvious that the base constructed in this way (it will be denoted as MB) is monotone. Also some other monotone bases can be defined which preserve the certain properties of the canonical base. The base MB preserves only one: the elements of both bases can be determined by only one pair of coordinates. The main difference is the following: the number of elements of canonical base whose composition is a matrix is equal to the value of one characteristic of the matrix - the number of units. For the monotone base this is not true.

3. Let us denote $NE_B(A)$ the number of elements of the base B whose composition mod 2 is the matrix A and by $||A||$ denote the number of units in matrix A. There holds $NE_{CB}(A) = ||A||$ for every $A \in \mathfrak{N}_N$. The distribution of the matrices from $\mathfrak{N}_N$ into the classes according the number of canonical base elements necessary to compose the matrix corresponds to distribution of the matrices into the classes with equal number of units. Since for every base there holds that the number of matrices A for which $NE_B(A) = k$ is equal to $C_{N^2}^k$, the cardinality of the classes according to NE_{MB} will be the same as for NE_{CB}. But in the case of monotone base in one class will belong the matrices due to some other properties and not the number of units. In this connection we can formulate the following problem. Let $\mathfrak{M}$ be a subclass of $\mathfrak{N}_N$; let us define $NE_B(\mathfrak{M})$ as $\max_{A \in \mathfrak{M}} NE_B(A)$. It is obvious that $NE_B(\mathfrak{N}_N) = N^2$ for arbitrary base B. The problem is, what is e.g. the

value of $NE_{MB}(E_N)$.

Theorem 1. $NE_{MB}(E_N) = 2N$.

Proof: Since the base MB is not invariant to rotation, the number of base elements will depend on two circumstatnces: first is the direction of the path which determines the corresponding matrix (definition 3), the second and more substancial is the number of changes of the path's direction (since the path is 4-connected only the change at angle $\pi/2$ is possible). It is because the change in direction determines either the unit of the matrix which has to be covered by an independent base element or the place from which the unsuitable removed units (zeroes) are to be added.

If we accept this and realize that to every change of the path's direction corresponds exactly one corner point of the matrix then we can formulate the following proposition.

Proposition 1. Let $A = (a_{ij})$ belong to E_N . If we determine the set C of all corner points of the matrix A and take the set D of all elements of the monotone base MB determined by the elements of C (every corner point with coordinates (i-d,j-d) determines - if possible - that element of the base MB which can be described by the pixel with coordinates (i,j)) and compose all elements in D with operation $\oplus$ then we obtain the matrix A .

Proof: Let us consider the monotone matrix A which fulfils the condition 1 of definition 3 and let $a_{ij} = 1$. We shall show that the number of base elements determined by the corner points of the matrix and covering the pixel (i,j) is odd and therefore after the composition the elements of monotone base will form in pixel (i,j) the value 1. It is sufficient to consider only the number of corner points in the rectangle (d,d), (d,j-d), (i-d,d) and (i-d,j-d) - because only these points determine the base elements which cover the pixel (i,j).

Since the number of corner points in the row is either zero or two we want to show there exists exactly one row which has one corner point inside and the second outside the rectangle. Let us assume that (s,j) ($s \leqslant i$) is the pixel with the

smallest first coordinate from all pixels in the j-th column which have the value 1. According to definition 3 this pixel must be separated from the zeroes by the points of the path defining the matrix A which lie in the row s-d . It is clear that in this row the path is changing its direction two times - once in the corner point with coordinates (s-d,k-d) ($k \leqslant j$) and twice in the corner point (s-d,m+d) ($m \geqslant j$) being outside the rectangle. Two such rows cannot exist according to given number of points in the path (which secures its monotone character). The similar proof can be used if $a_{ij} = 0$ and for the matrices rotated at $\pi/2, \pi, 3\pi/2$.

Now back to the proof of theorem 1. From the relation between the corner points and the elements of the monotone base one can conclude that the maximal number of the base elements is needed to compose a monotone matrix with the maximal number of corner points. It is shown below that this is not entirely true. The maximal number of corner points occur in such matrix which has units on the diagonal. This matrix contains 2N + 2 corner points (2N + 1 on the diagonal and one in the opposite corner). It is possible to show that for every rotation at multiples of $\pi/2$ always at least three corner points determine no base element. For this matrices it holds $NE_{MB} \leqslant \leqslant 2N - 1$.

The key is to find a matrix with almost maximal number of corner points which all will determine the corresponding base element. The example of such matrix is in Fig. 2 . This

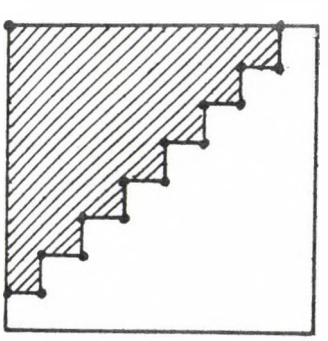


Fig.2

matrix has units on the diagonal left to secondary diagonal. The number of corner points is 2N-1 on the diagonal, one cor-

ner point in the opposite corner all determining the elements of monotone base. This completes the proof.

4. The positive consequence of theorem 1 is the possibility of data compresion at the storage of monotone matrices. In fact the storage of concrete base element requires only 2 log N bits so that at most 4N log N bits are sufficient for the storage of arbitrary monotone matrix.

The second consequence of the proved theorem is the fact that the structural properties were discovered (first for the case of monotone matrices) according to which the matrices can be arranged in the classes with the same number of base elements whose composition produces exactly those matrices. It was shown that the property is the number of corner points in the matrix which is in good correlation with the number of base elements referred. One can conjecture this true also in the general case. The following proposition approves that conjecture.

Proposition 2. Proposition 1 holds not only for matrix $A \in \mathcal{E}_N$, but for arbitrary matrix $A \in \mathcal{M}_N$.

Proof: Let us assume again $a_{ij} = 1$. We have to show that the number of corner points in the rectangle examined in proposition 1 is odd. In this case there holds that in each row and each column the number of corner points is even (including of course zero). The concept of the path is replaced by the concept of the border of an object (represented also by the points of the p-grid).

We shall first examine the pixels in the column j with the first coordinate less than i in descending order. If we come to first zero pixel in the row s then this must be separated from units by the points of the border in the row $s+d$. The border in this row "crosses" the side of the rectangle having one of its corner points inside and one outside the rectangle. Every next combination of units and zeroes must be separated twice having two corner points inside. So the number of corner points corresponding to column is odd.

Now we take the pixels in the row i with the second

coordinate less than j in descending order. If the value of pixel $(i,j-1)$ is equal to zero it is separated by border whose corner point inside was already counted. Every next combination of units and zeroes must be separated twice having two corner points inside the rectangle. If $a_{i,j-1} = 1$ then we search for the first zero which must be again separated. The combination of units and zeroes are separated by even number of corner points but one of them could not be counted because it would be counted twice (in row and also in column examination). So the number of corner points corresponding to row i is even and the total number of corner points is odd. The proof is completed.

The assertion of proposition 2 is in good agreement with the results of Haverlík [4] who showed that if only corner points of the matrix are given the whole matrix can be unambigously reconstructed. (His procedure is based on the examination of the parity of the number of corner points which cover the corresponding pixel of the matrix). The algebraic substance of this procedure is shown in the proof of proposition 2.

5. In this contribution the unusual relation between the corner points of binary image and the elements of monotone base for vector space of binary images including its consequences was shown. This suggests that an algebraic view on binary images can deep our knowledge about the features and characteristic needen in picture processing.

References

[1] Duda R.O. and Hart P.E.: Pattern classification and scene analysis, Wiley 1973

[2] Ftáčnik M.: The complexity characteristic of some Boolean models of images. to appear in Acta Mathematica U.C.,1986

[3] Ftáčnik M.: The complexity and recognition of some classes of Boolean matrices with the use of monotonicity, the manuscript of CSc. thesis, Bratislava 1986

[4] Haverlík I.: The synthesis of logical networks for the realization of Boolean matrices with the given number of corner points, In: Discrete mathematics PWN Warsaw 1978

[5] Rosenfeld A.: Connectivity in Digital Pictures, J. of the ACM, Vol. 17 (1970), 146-160

Proc. IMYCS '86 October 13-17, 1986
Smolenice Castle, ČSSR

POSITIVE RELATIVIZATIONS OF THE HAUSDORFF HIERARCHY GENERATED BY NP

Th. Gundermann

Department of Mathematics
Friedrich-Schiller University
Jena
6900
German Democratic Republic

1. Introduction

Recently the Boolean closure of NP (denoted by BC(NP)) has been studied from several points of views. Since F. Hausdorff [1] the Boolean closure of a class K of sets closed under union and intersection is known to be the union of classes which we call the Hausdorff hierarchy generated by K. In the case of K = NP we have $D_1 = NP$, $C_1 = coNP$,

$$D_{n+1} = C_n \vee NP = \{X \cup Y : X \in C_n \wedge Y \in NP\} ,$$
$$C_{n+1} = D_n \wedge coNP = \{X \cap Y : X \in D_n \wedge Y \in coNP\} ,$$

$\mathcal{H}(NP) = \{C_n : n \in \mathbb{N}\} \cup \{D_n : n \in \mathbb{N}\}$. Then according to Hausdorff:
Fact 1: $BC(NP) = \bigcup_{n=1}^{\infty} C_n$.

In Wechsung [1] and Wechsung, Wagner [1] new acceptance types for nondeterministic Turing machines have been introduced in such a way that the polynomial time complexity classes with respect to these new notations are exactly the classes of $\mathcal{H}$ (NP). Related questions concerning BC(NP) are investigated in Köbler, Schöning [1] , Papadimitriou, Yannakakis [1] , Wagner [1] .

The complexity classification of problems defined by restricting NP-complete problems to those instances having unique solutions requires finer hierarchies within BC(NP). The class 1NP of sets which can be accepted by nondeterministic

polynomial time Turing machines by exactly one acceting path is unlikely closed under union ($1NP \overset{?}{\subset} 1NP \vee 1NP$). This motivates examining the Hausdorff hierarchy generated by 1NP in Gundermann,Wechsung [2] . We define

$A_1 = 1NP$, $A_{i+1} = A_i \vee 1NP$, $A_{i_1 \ldots i_s} = A_{i_1} \wedge \ldots \wedge A_{i_s}$,

$\mathcal{H}(1NP) = \{ A_{i_1 \ldots i_s} : s, i_1, \ldots, i_s \in \mathbb{N} \}$.

The question whether the hierarchies $\mathcal{H}(NP)$, $\mathcal{H}(1NP)$ are strict is at least as difficult as the P-NP-problem. It is known which relationships between the classes of the hierarchies are possible under suitable relativizations. In Gundermann,Wechsung [1] we showed that for $\mathcal{H}(NP)$ everything is possible:

Fact 2: There exists a recursive oracle A such that

$$C_1^A \subset C_2^A \subset \ldots \subset C_k^A \subset \ldots .$$

Fact 3: For every $k \geq 1$ there exists a recursive oracle A such that $C_1^A \subset C_2^A \subset \ldots \subset C_k^A = C_{k+1}^A = \ldots$.

In Gundermann,Wechsung[2] similar results for $\mathcal{H}(1NP)$ are proved. To formulate them we need the following concepts:

level $A_{i_1 \ldots i_s} = 2(i_1 + \ldots + i_s) - 2s + 3$ and

index $A_{i_1 \ldots i_s}$ = index $A_{i_{p(1)} \ldots i_{p(s)}}$ for every permutation p of $\{1,2,\ldots,s\}$ and

index $A_{i_1 \ldots i_s} = (i_1, \ldots, i_s)$ for $i_1 \leq i_2 \leq \ldots \leq i_s$.

Fact 4: If $X, Y \in \mathcal{H}(1NP)$ and level X < level Y then there exists a recursive oracle B such that $Y^B \not\subseteq X^B$.

Fact 5: If $X, Y \in \mathcal{H}(1NP)$ and level X = level Y but index X ≠ index Y then there exists a recursive oracle B such that $Y^B \not\subseteq X^B$ and $X^B \not\subseteq Y^B$.

We know that finding an oracle which separates two complexity classes need not be an argument that those two classes must be different in the unrelativized case. But it is also useful to look for restrictions on the oracle machines which bound the power of the oracle in such a way that separation in the relativised case implies the inequality of the unrelativized one. Investigations in such a direction for questions concerning the relationships between P, NP, coNP, PSPACE may be found for instance in Book,Long,Selman [1,2] .

In Section 3 we develop "positive relativizations" of questions $\mathfrak{A}NP \overset{?}{\subseteq} \mathfrak{B}NP$ for arbitrary acceptance types $\mathfrak{A}$ and $\mathfrak{B}$;

that is we restrict the behaviour of these oracle machines to obtain statements such as $\mathfrak{A}NP \subseteq \mathfrak{B}NP \leftrightarrow \bigwedge_{B \subseteq \Sigma^*} \mathfrak{A}NP_R^B \subseteq \mathfrak{B}NP_R^B$ where $\mathfrak{A}NP_R^B$ is the class of languages L such that $L \in \mathfrak{A}NP^B$ is witnessed by a nondeterministic polynomial time bounded oracle machine operating with restriction R. Our result can be improved for acceptance types characterising classes of $\mathfrak{K}(NP)$. Using the method of the proofs of Fact 2 and Fact 4 we are able to show that the restriction for questions concerning $\mathfrak{K}(NP)$ is as weak as possible.

2. Basic concepts

We always use an input alphabet with two letters $\Sigma = \{0,1\}$. To describe our classes we provide the following acceptance notations. Definition :

1. $\mathfrak{A}$ is called an acceptance type if there is a natural number k such that $\mathfrak{A} \subseteq \mathbb{N}^k$ where $\mathbb{N}$ is the set of natural numbers $\{0,1,\ldots\}$

2. Let $M^{(\,)}$ be a nondeterministic oracle machine having the set $\{S_0,S_1,\ldots,S_k\}$ of final states and $A \subseteq \Sigma^*$.
$Leaf_M^i A\,(x)$ = number of paths of $M^A(x)$ reaching a final configuration with state S_i.
$Leaf_M A\,(x) = (Leaf_M^1 A\,(x),\ldots,Leaf_M^k A\,(x))$ is called the leaf number vector of M on x relative to the oracle A.

3. M^A $\mathfrak{A}$-accepts $x \longleftrightarrow Leaf_M A\,(x) \in \mathfrak{A}$, provided $M^{(\,)}$ has a set of k+1 final states (S_0 is needed for waste).

4. $L_{\mathfrak{A}}(M^A) = \{\, x : M^A \ \mathfrak{A}\text{-accepts } x \,\}$.

5. $\mathfrak{A}NP^A = \{\, L_{\mathfrak{A}}(M^A) : M^{(\,)}$ is a nondeterministic oracle machine working in polynomial time$\,\}$.

If $A = \emptyset$ we write for short $M^\emptyset = M$, $\mathfrak{A}NP^\emptyset = \mathfrak{A}NP$.
The classes of $\mathfrak{K}(NP)$ and $\mathfrak{K}(1NP)$ are certain $\mathfrak{A}NP$. We have for instance $NP \vee coNP = \{(m,n) : m,n \in \mathbb{N} \wedge (m \neq 0 \vee n = 0)\}\,NP$,
$1NP = \{1\}NP$.

In a similar way we define classes of functions.

2'. Let $T^{(\,)}$ be a nondeterministic transducer with oracle having the set $\{S_0,\ldots,S_k\}$ of final states.
$Leaf_T^i A\,(x,y)$ = number of paths of $T^A(x)$ reaching final state S_i and computing y. $Leaf_T A\,(x,y)$ is defined like above.

3'. T^A $\mathfrak{A}$-computes $(x,y) \longleftrightarrow Leaf_T A\,(x,y) \in \mathfrak{A}$.

A multivalued partial function $f \subseteq \Sigma^* \times \mathfrak{P}(\Sigma^*)$ is said to be

$\mathfrak{A}$-computable in polynomial time if there is a transducer T with polynomial clock such that $y \in f(x) \leftrightarrow$ T $\mathfrak{A}$-computes (x,y).

4'. $f_{\mathfrak{A}}(T^A) = \{(x,Y) : Y \subseteq \Sigma^* \wedge \bigwedge_{y \in Y} T^A \mathfrak{A}\text{-computes } (x,y)\}$.

5'. $\mathfrak{A}NP^AMV = \{f : f = f_{\mathfrak{A}}(T^A)$ and $T^{(\ )}$ is a nondeterministic polynomial time bounded transducer with oracle $\}$.

For $f \subseteq \Sigma^* \times \mathfrak{P}(\Sigma^*)$ let $f_o = \{(x,y) : y \in f(x)\}$. We say f_o has polynomial bounded size iff there is a polynomial p such that

$$\bigwedge_{x,y} \; y \in f(x) \longrightarrow |y| \leq p(|x|) .$$

For every set L let X_L be the function defined by

$$X_L(x) = \begin{cases} 1 \text{ , if } x \in L, \\ \text{undefined otherwise.} \end{cases}$$

Then we have the following correspondence between complexity classes for languages and functions:

Theorem 6: 1. $L \in \mathfrak{A}NP^A \longleftrightarrow X_L \in \mathfrak{A}NP^AMV$

2. $f \in \mathfrak{A}NP^AMV \longleftrightarrow f_o \in \mathfrak{A}NP^A$ and f_o has polynomial bounded size.

Theorem 7: Let $\mathfrak{A}$, $\mathfrak{B}$ be acceptance types. Then

$$\mathfrak{A}NP^A \subseteq \mathfrak{B}NP^A \longleftrightarrow \mathfrak{A}NP^AMV \subseteq \mathfrak{B}NP^AMV .$$

3. Positive Relativizations

First we give a positive relativization for questions concerning arbitrary acceptance types.

Definition: A nondeterministic Turing machine with oracle $M^{(\ )}$ is called confluent iff for any input x and any oracle A there is a set $\{y_1,\ldots,y_l\}$ of words $y_i \in \Sigma^*$ such that on every path of the computation tree $M^A(x)$ the oracle is queried exactly about $y_1,\ldots,y_l$. (Note that this concept is more extensive than the Definition in Book,Long,Selman [1])

$\mathfrak{A}NPC^A = \{L_{\mathfrak{A}}(M^A) : M^{(\ )}$ is a confluent nondeterministic oracle machine working in polynomial time$\}$.

Theorem 8: Let $\mathfrak{A}$, $\mathfrak{B}$ be acceptance types. Then

$$\mathfrak{A}NP \subseteq \mathfrak{B}NP \longleftrightarrow \bigwedge_{A \subseteq \Sigma^*} \mathfrak{A}NPC^A \subseteq \mathfrak{B}NP^A . \blacksquare$$

For the classes of $\mathfrak{K}(NP)$ the following property can be used to improve the above result.

Lemma 9: For any $X \in \mathfrak{K}(NP)$ there exists a chain respecting acceptance type $\mathfrak{A} \subseteq \mathfrak{P}\mathbb{N}^{k(X)}$ such that $\{x_1,\ldots,x_k\} \in \mathfrak{A}$ and $x_i \neq 0$ implies $x_{i-1} \neq 0$ and $X = \mathfrak{A}NP$. (Wechsung [1]) $\square$

Lemma 10: If $\mathfrak{A}NP \in \mathfrak{K}(NP)$ then there exists a chain respecting

acceptance type $\mathfrak{B}$ such that $\mathfrak{A}\,\mathrm{NPMV} = \mathfrak{B}\,\mathrm{NPMV}$.□

Now we shall define a weaker restriction on nondeterministic oracle machines than confluence.

Definition: Let $M^{(\,)}$ be a nondeterministic oracle machine. For any set A and any input x let $Q(M,A,x)$ be the set of strings y such that there is a path in $M^A(x)$ on which the oracle is queried about y. $Q(M,x) = \bigcup_A Q(M,A,x)$.

$\mathfrak{A}\,\mathrm{NP}_B^A = \{ L_{\mathfrak{A}}(M^A) : M^{(\,)}$ is an nondeterministic oracle machine working in polynomial time and there is a polynomial p such that $\mathrm{card}\, Q(M,x) \leq p(|x|) \}$.

Note that $\mathfrak{A}\mathrm{NPC}^A \subseteq \mathfrak{A}\mathrm{NP}_B^A$.

Theorem 11: Let $\mathfrak{A}$, $\mathfrak{B}$ be acceptance types characterizing classes of $\mathcal{H}(\mathrm{NP})$. Then $\mathfrak{A}\,\mathrm{NP} \subseteq \mathfrak{B}\,\mathrm{NP} \longleftrightarrow \bigwedge_A \mathfrak{A}\mathrm{NP}_B^A \subseteq \mathfrak{B}\mathrm{NP}^A$.

Sketch of proof: $\longleftarrow$ Obvious, because $\mathfrak{A}\mathrm{NP} = \mathfrak{A}\mathrm{NP}_B^{\emptyset}$.

$\longrightarrow$. For $c,d \notin \Sigma$ let $\mathrm{code}(x_1,\ldots,x_n)$ be a string coding the finite set $\{x_1,\ldots,x_n\}$ of words belonging to $(\Sigma \cup \{c,d\})^*$. Assume that code and also its inverse are computable in polynomial time. Then we can use

Lemma 12: For any nondeterministic Turing machine $M^{(\,)}$ working in polynomial time p and having the set of final states $\{S_0,\ldots,S_k\}$ there exists a nondeterministic transducer M' working in polynomial time p' and having the set of final states $\{S_0,\ldots,S_k\}$ in such a way that for any oracle A and any x

$\mathrm{leaf}^i_{M'A}(x) = \mathrm{card}\{ b : b$ is a path in $M'(x)$ on which the final state S_i is reached and for the word $w = \mathrm{code}(w_1,\ldots,w_t)$ computed on b holds that $U = \{u : cu \in \{w_1,\ldots,w_t\}\} \subseteq A$ and $V = \{v : dv \in \{w_1,\ldots,w_t\}\} \subseteq \bar{A}\}$

for $1 \leq i \leq k$.□

We call M' a speculating machine equivalent to $M^{(\,)}$.

In general we cannot force that a NP_B-machine is confluent but we can construct equivalent speculating machines which compute codes of sets of equal cardinality.

Lemma 13: For any speculating machine M working in polynomial time p and any polynomial q with $p(n) \leq q(n)$ there exists a speculating machine M' working in polynomial time such that for any x and any set W' with $\mathrm{card}\, W' = q(|x|)$:

$\mathrm{card}\{b : b$ is a path in $M(x)$ leading to the final state S_i

and computing code W with $W \subseteq W'\} = \text{card}\{b : b$ is a path in $M'(x)$ leading to S_i and computing code $W'\}$ for $1 \leq i \leq k$.□

Now let $L \in \mathfrak{A}NP_B^A$ be witnessed by M^A. Let p be a polynomial clock for $M^{(\,)}$ and $q(|x|)$ a polynomial bounding the cardinality of $Q(M,x)$. According to the preceding two lemmas there is a speculating machine M' such that for any x,W

$$(1)\quad \begin{cases} \{cw : w \in A \cap Q(M,x)\} \cup \{dw : w \in \bar{A} \cap Q(M,x)\} \subseteq W \subseteq \\ \quad \subseteq \{cw : w \in A\} \cup \{dw : w \in \bar{A}\} \\ \text{and card } W = q(|x|) \text{ and } y \in W \rightarrow |y| \leq p(|x|) \end{cases}$$

implies

$$\text{leaf}_M^i A\,(x) = \text{card}\{b : b \text{ is a path in } M'(x) \text{ leading to } S_i \text{ and computing code } W\} \text{ for } 1 \leq i \leq k .$$

This means : x is $\mathfrak{A}$-accepted by $M^A \longleftrightarrow$ (x,code W) is $\mathfrak{A}$-computed by M' for any W satisfying (1). According to Theorem 7 we have the existende of a machine M" working in polynomial time and $f_{\mathfrak{A}}(M') = f_{\mathfrak{L}}(M")$, (x,code W) is $\mathfrak{A}$-computable by M' $\longleftrightarrow$ (x,code W) is $\mathfrak{L}$-computable by M".

Now there exists an oracle machine $M'''^{(\,)}$ equivalent to M". (M''' first simulates M" and asks then the oracle about the words in code W).We may assume that $\mathfrak{L}$ is chain respecting. Then there is a W_{max} satisfying (1) such that for all W with (1): If code W is computed on path to S_i by M" on input x then there is a path to S_j for at least one $j \geq i$ on which code W_{max} is computed.

Let j_o be the maximal number j such that code W_{max} is computed on a path to S_j. Then M'''^A reaches S_{j_o} on input x but no S_j with $j_o < j$. Thus, if W_{max} is $\mathfrak{L}$-computed by M" then x is $\mathfrak{L}$-accepted by M'''^A. ■

The restriction given above is as weak as possible. If the machines are allowed to ask the oracle more then polynomially many different words, then those machines are to powerful for positive relativations. We give a precise formulation:

Definition: A function f from $\mathbb{N}$ into $\mathbb{N}$ is said to be super-polynomial if there is no polynomial p such that $\bigwedge_n f(n) \leq p(n)$

$$\mathfrak{A}NP_f^A = \{L_{\mathfrak{A}}(M^A) : M \text{ is an oracle machine working in polynomial}$$

time and card $Q(M,x) \leq f(|x|)$ for all x .

A numbertheoretical function f is computable in polynomial time iff $F: 1^n \rightarrow$ bin f(n) is computable in p(n) steps for some polynomial p.

Theorem 14: For any superpolynomial f which is computable in polynomial time there is a recursive oracle A such that

$$C_{1_f}^A \subset C_{2_f}^A \subset \ldots \subset C_{k_f}^A \subset \ldots .$$

The method to prove this Theorem has already been used in Gundermann,Wechsung [1,2].■

I want to thank Gerd Wechsung for his supports and many hints.

Literature

Book, R.V., Long, T.J., Selman, A.L.

1 Quantitative Relativizations of Complexity classes, SIAM J. COMP. 13(1984), 461-487

2 Qualitative Relativizations of Complexity classes, Theoret. Comput. Sci 26(1985)

Gundermann, Th., Wechsung, G.

1 Relativizing the Hausdorff hierarchy generated by NP, to appear

2 Nondeterministic Turing machines with modified acceptance, to appear

3 Counting Classes with Finite Acceptance Types, to appear

Hausdorff, F.

1 Grundzüge der Mengenlehre, Leipzig 1914

Köbler, J., Schöning, U.

1 The difference and truth-table hierarchies for NP, submitted to RAIRO Inf. theor.

Papadimitriou, Ch.H., Yannakakis, M.

1 The complexity of facets (and some facets of complexity), JCSS 28(1984), 244-259

Wagner, K.

1 More complicated questions involving about Maxima and Minima and some closures of NP, to appear

Wechsung, G.

1 On the Boolean closure of NP, LNCS 199, 485-493

Wechsung, G., Wagner, K.

1 On the Boolean closure of NP, submitted to JCSS.

Proc.IMYCS '86 October 13-17,1986
Smolenice Castle,ČSSR

REAL TIME COMPUTATION BY HOMOGENEOUS STRUCTURES

I.JANETKA

Department of Theoretical Cybernetics
Faculty of Mathematics and Physics
Comenius University
842 15 Bratislava
Czechoslovakia

ABSTRACT

An 1-dimensional homogeneous structure is formally introduced as a real-time acceptor. The computational characteristics of 1-dimensional homogeneous structures are illuminated by establishing severel results concerning the sets of Boolean functions that they recognize. This paper proves that the set of symmetric Boolean functions is real-time acceptable by 1-dimensional homogeneous structures.

1.Introduction

The paper deals with the real-time computability on homogeneous structures in the sense introduced by Cole [1]. The homogeneous structure is a formalization of the concept of an infinite regular array of identical finite-state machine uniformly interconnected in the sense that each machine can directly receive information by means of interconnecting wires from a finite number of neighbouring machines where the spatial arrangement of these neighbouring machines is the same relative to each machine in the array. Each machine can synchronously change its state at discrete time steps as a function of the states of the neighbouring machines, but will be identical for each machine in the array at any given time step. An 1-dimensional homogeneous structure can be imagined

as an infinite tape of identical cells A_i, i=...,-2,-1,0,1,2, ...

In order to study the computation by 1-dimensional homogeneous structures, it is possible to have a special cell A_0 as the input and output cell. The subclass of 1-dimensional homogeneous structures we shall deal with are those that compute characteristic functions (functions with range 0,1). To each characteristic function f there corresponds the set A_f of all objects n, such that f(n)=1. Homogeneous structures that compute characteristic functions are called "acceptors", since one attributes to them task of accepting the objects in A_f by producing the output 1 in the output cell and rejecting the objects not in A_f by producing the output 0 as the state in the output cell. All cells of 1-dimensional homogeneous structures are divided into two parts of 1-dimensional homogeneous structures by cell A_0. All cells A_{-1}, A_{-2}, ..., A_{-i} in the left part are those that contain a code of the object from the set A_f at the 0-th step of computation. The computation is realized in the right part. Let the inputs in cells A_{-1}, A_{-2}, ..., are fed in from right to left and transfered from left to right to input cell A_0. A real-time acceptor must generate the output at the i-th step of computation whereby an information about the code of the input object in cell A_{-k}, k=1, 2, ..., i, is received at the k-th step of computation in cell A_0.

Cole [1] has shown that they can recognize the set of all palindromes and the set of all strings of the form xx. Fischer [2] has shown that 1-dimensional arrays can generate the characteristic sequence of the set of all prime numbers, i.e., the cell A_0 will put out a 1 at the time t when t is a prime and a 0 at the time t otherwise.

Let the inputs be written in binary notation and represent the values of the Boolean function; all values of n-ary Boolean function are written in cells A_{-1}, A_{-2}, ..., A_{-2^n}, n=1, 2, ... We show that 1-dimensional arrays can recognize the set all symetric Boolean functions in real time, i.e., the cell A_0 will put out a 1 at the time 2^n, if A_{-1}, ..., A_{-2^n} are the

values of the symetric Boolean function at the time 0, and a 0 at the time 2^n otherwise.

The paper is organized as follows. In section two the necessary definitions are given. The third section contains the main results.

2. Basic Notions

2.1. The letters Z, N will denote the sets of integers and nonnegative integers, respectively.

D e f i n i t i o n 2.2. An 1-dimensional homogeneous structure (1-HS) is an ordered quadruple $\langle Z,E,V,F\rangle$, where

(1) Z represents an 1-dimensional regular array,

(2) $E = \{0, 1, \ldots, r-1\}$ is the set of states of the 1-HS,

(3) V is the neighbourhood index of the 1-HS,

(4) F is the local transition function of the 1-HS, $F : E^n \longrightarrow E$,

(5) n is the number of neighbouring machines in the 1-HS.

The n neighbours of cell z ($z \in Z$) are cells $z+a_1$, $z+a_2$, ..., $z+a_n$, where $V = (a_1, a_2, \ldots, a_n)$.

The elementary operation of an 1-HS is the simultaneous application of a local transformation F to the neighbourhood of every cell of the 1-dimensional regular array while a local transformation F produced the next state of an individual cell z in terms of the states of the machines which are neighbours of the cell z. The action performed by one such elementary operation is called "step".

D e f i n i t i o n 2.3. Let C_E denotes the set of all configurations with respect to Z and E, i.e., C_E is the set of all mapping $Z \longrightarrow E$. Let the image of $z \in Z$ under $c \in C_E$ is written c(z). The neighbourhood index V and a map $F : E^n \longrightarrow E$ define a global map $G : C_E \longrightarrow C_E$ as follows. $G(c) = c'$ if and only if for any $z \in Z$, $c'(z) = F(c(z+a_1), c(z+a_2), \ldots, c(z+a_n))$. The action of applying a map G to the homogeneous space is

called step of 1-HS.

D e f i n i t i o n 2.4. A set of Boolean functions A_f is said to be real-time acceptable if there exist 1-HS such that

(1) $V = (-1, 0, 1)$,

(2) Let $u_0, u_1, \ldots, u_{2^n-1}$ be the values of Boolean function f of n variables such that

$$u_i = f(x_1,\ldots,x_n),\ i = x_1 2^0 + \ldots + x_n 2^{n-1},$$

then the input data $u_0, u_1, \ldots, u_{2^n-1}$ are put into cells $A_{-1}, A_{-2}, \ldots, A_{-2^n}$, respectively,

(3) The cell A_0 will put out a 1 at the time 2^n when the function f is element of A_f and cell A_0 will put out a 0 at the time 2^n otherwise.

D e f i n i t i o n 2.5. A Boolean function $f: A^n \to A$ (where A denotes a set $\{0,1\}$), $n \in N$, is called symmetric if for all permutations π of $\{1,2,\ldots,n\}$

$$f(c_1,\ldots,c_n) = f(c_{\pi(1)},\ldots,c_{\pi(n)}).$$

D e f i n i t i o n 2.6. A Boolean function $f: A^n \to A$ is called monotone if for all $c, d \in A^n$, $c = (c_1, \ldots, c_n)$, $d = (d_1, \ldots, d_n)$ and $c_1 \leq d_1, \ldots, c_n \leq d_n$

$$f(c_1,\ldots,c_n) \leq f(d_1,\ldots,d_n).$$

D e f i n i t i o n 2.7. A Boolean function $f: A^n \to A$ is called linear if for all $c \in A^n$, $c = (c_1, \ldots, c_n)$

$$f(c_1,\ldots,c_n) = a_0 + a_1c_1 + \ldots + a_nc_n \pmod 2$$

where $a_i \in \{0,1\}$, $i = 1,\ldots,n$.

D e f i n i t i o n 2.8. A Boolean function $f: A^n \to A$ is called self-dual if for all $c \in A^n$, $c = (c_1, \ldots, c_n)$

$$f(c_1,\ldots,c_n) = \bar{f}(\bar{c}_1,\ldots,\bar{c}_n)$$

where $\bar{x}$ means the opposite of x.

D e f i n i t i o n 2.9. A Boolean function $f: A^n \to A$ is called "function preserving a 0" if $f(0,\ldots,0) = 0$.

3. Main Results.

The proofs of our main results are quite long containg

many lemmas from which we present the most interesting and fundamental ones.

L e m m a 3.1. Let $u_0, \ldots, u_{2^{n+1}-1}$ be the values of arbitrary $n+1$ -ary symetric function f such that

$$u_i = f(x_1, \ldots, x_{n+1}), \quad i = x_1 2^0 + \ldots + x_{n+1} 2^n.$$

Then

$$\begin{aligned}
&u_{2^n-1} = u_{2^n} \\
&u_{2^n-1+1} = u_{2^n+1} \\
&\cdots \quad \cdots \\
&\cdots \quad \cdots \\
&\cdots \quad \cdots \\
&u_{2^n-1} = u_{2^n+2^{n-1}-1}
\end{aligned} \tag{3.1.1}$$

$$\begin{aligned}
&u_{2^n-1+2^{n-2}} = u_{2^n+2^{n-1}} \\
&u_{2^n-1+2^{n-2}+1} = u_{2^n+2^{n-1}+1} \\
&\cdots \quad \cdots \\
&\cdots \quad \cdots \\
&\cdots \quad \cdots \\
&u_{2^n-1} = u_{2^n+2^{n-1}+2^{n-2}-1}
\end{aligned} \tag{3.1.2}$$

$$\begin{aligned}
&\cdots \\
&\cdots \\
&\cdots \\
&u_{2^n-1+2^{n-2}+\ldots+2^{n-k}} = u_{2^n+2^{n-1}+\ldots+2^{n-k+1}} \\
&u_{2^n-1+2^{n-2}+\ldots+2^{n-k}+1} = u_{2^n+2^{n-1}+\ldots+2^{n-k+1}+1} \\
&\cdots \quad \cdots \\
&\cdots \quad \cdots \\
&\cdots \quad \cdots \\
&u_{2^n-1} = u_{2^n+2^{n-1}+\ldots+2^{n-k}-1}
\end{aligned} \tag{3.1.k}$$

for every $k \in \{3, \ldots, n-1\}$.

P r o o f. Since $f(x_1, \ldots, x_{k-1}, 0, x_{k+1}, \ldots, x_n, 1) = f(x_1, \ldots, x_{k-1}, 1, x_{k+1}, \ldots, x_n, 0)$ for every $k \in \{1, \ldots, n\}$, one can easily verify that (3.1.1), (3.1.2), ..., (3.1.k) holds.

N o t e 3.2. The previous lemma is used for evaluting the values $u_{2^n}, u_{2^n+1}, \ldots, u_{2^{n+1}-2}$ of arbitrary $n+1$-ary symmetric function f if we know the values $u_0, u_1, \ldots, u_{2^n-1}$.

T h e o r e m 3.3. Let A_{f_1} be a set of symmetric Boolean functions. Then A_{f_1} is real-time acceptable.

P r o o f. It is easy to construct the 1-HS if n = 1. Let us assume, therefore, that $n \geq 2$. By definition 2.4. the input data $u_0, \ldots, u_{2n-1}$ are put into cells $A_{-1}, \ldots, A_{-2n}$, respectively at the time 0. The machines in 1-HS D consist of eight registers A, B, ..., H. 1-HS D processes according to the following steps:

1°. Symbols u_1, u_2 are stored in registers A of the cells A_1, A_2 respectively. Let i = 1, j = 2.

2°. Second half of the input symbols stored in registers A of the cells A_i, $A_{i+1}, \ldots, A_j$ is synchronously propagated trough the C register of the array to the right and put into cells $A_{j+1}, A_{j+2}, \ldots, A_{j+\lceil \frac{j-i}{2} \rceil}$.

3°. If $j+1 = j+\lceil \frac{j-i}{2} \rceil$ then go to step 4°.

4°. The contents of registers A of the cells $A_1, A_2, \ldots, A_{j+1}$ are put into registers E. This process is executed synchronously in cells $A_1, A_2, \ldots, A_{j+1}$.

5°. Put i:=1 and go to step 2°.

Registers G, H are used for synchronization of the process in step 2°, 4°, respectively. When registers E are feed their contents is propagated trough the array to the left. The contents of the cell A_{-1} and register A of the cell A_0 are compared. According to lemma 3.1. and the note 3.2. if $u_0, \ldots, u_{2n-1}$ are the values of symmetric function than the contents the registers of compared will be equal. Only the value u_{2k-1}, k = 1, 2, ..., does not depend on the previous values so this value is propagated trough the F registers of the array to the right.

A precise proof of theorem 3.3. is given by Janetka [3].

Using the idea of evaluting the values before their putting to the cell A_0 we have obtained the following results.

T h e o r e m 3.4. Let A_{f_2} be a set of monotone Boolean functions. Then A_{f_2} is real-time acceptable.

T h e o r e m 3.5. Let A_{f_3} be a set of linear Boolean functions. Then A_{f_3} is real-time acceptable.

T h e o r e m 3.6. Let A_{f_4} be a set of self-dual Boolean functions. Then A_{f_4} is real-time acceptable.

T h e o r e m 3.7. Let A_{f_5} be a set of Boolean functions preserving 0. Then A_{f_5} is real-time acceptable.

A proof of theorem 3.4. - 3.7. is given by Janetka [3].

As a corollary of theorem 3.4. - 3.7. we get

C o r o l l a r y 3.8. Let A_f be a set of complete systems of Boolean functions in the sence introduced by Post[4]. Then A_f is real-time acceptable.

REFERENCE

[1] Cole S.N., Real-time computation by n-dimensional iterative arrays of finite-state machines. IEEE trans. on computers, vol.C-18, No.4, 1969.

[2] Fischer P., Generation of primes by a one-dimensional real-time iterative array. J.ACM, 12, No.3(1965).

[3] Janetka I., Computational complexity on the homogeneous structures. CSc.dissertation. Comenius University, Bratislava, 1986.

[4] Post E., Introduction to a general theory of elementary propositions. Amer.J.Math. 43(1921).

Proc. IMYCS '86 October 13-17, 1986
Smolenice Castle, ČSSR

ORDERED SETS, COMPARISONS AND GRAPHS OF PERMUTATIONS

Ľubor Kollár

Institute of Applied Mathematics
Comenius University
842 15 Bratislava
Czechoslovakia

Abstract: A new data model of comparison based sorting like procedures is presented. Using the model we give an $O(n^4)$ algorithm with an output - optimal algorithm for merging 2 and n element linearly ordered sets, we pose a transparent formulation of one open problem on sorting, and an equivalence of a so called parity problem and a sorting problem is shown using a very simple argument.

1. Introduction

A process of a comparison based sorting can be viewed in different ways. A standard model of a sorting algorithm is a comparison tree /Kn/ which reflects gradual enrichments of an order until a total order is obtained in a leaf of the tree. But the same process can be viewed also as a sequence of splits of a set of all admissible total orders (similarly it is done in /KS/). For example, sorting an n element set we start with n! possible total orders (permutations), the first comparison splits this set into two (nonintersecting) subsets; it depends on the result of the comparison which of these sets is used in the following similar process. The process ends when a singleton permutation is obtained. Each of the sets of permutations in this process is associated with some partially ordered set (poset).

Many different models of posets have been used up to now (the best known is the Hasse diagram) but non of them clearly refle-cts the interconnection between a comparison and corresponding partition of a set of permutations.

In the following section we introduce a graph model with permutations for nodes and a simple rule for edges. It is shown how to check if such a graph corresponds to a poset or not (Characterization theorem) and which partition of this graph corresponds to a comparison and which not (Splitting theorem). In Section 3 we show how the model can be used in solving some problems on comparison based sorting like procedures.

2. Graphs of permutations

We denote $\mathcal{P}$ a set of all permutations of m element base set M (only finite sets are considered throughout the paper) and $\mathcal{L}$ a set of all linear orderings of M. The simple one-to-one correspondence between the elements of $\mathcal{P}$ and $\mathcal{L}$ involves us to use terms linear ordering and permutation interchangeably.

Two linear orderings l_1, l_2 from $\mathcal{L}$ are called neighboring iff $|l_1 \setminus l_2| = 1$. Using permutations instead of linear orderings it is easy to see that two permutations from $\mathcal{P}$ are neighboring (i.e. they correspond to neighboring linear orderings) iff one can be obtained from the other by reversing the order of one pair of adjacent elements. This symmetric relation on a set $P \subseteq \mathcal{P}$ of permutations will be denoted by $N(P)$.

The following definition gives a feasible model of the neighboring relation.

Definition 1. Let P be a set of permutations. A graph of P is an unoriented graph $G(P) = (P, N(P))$. Such graph is called a graph of permutations.

Particularly important is the graph $G(\mathcal{P})$ of all permutations of M. This graph has m! nodes; Figure 1 shows the graph for m=3. Such graphs were briefly mentioned in /Kn/ but not in the context of posets as it is in our paper.

To give a better understanding of how the graphs look like, we list some easily provable properties of the graph $G(\mathcal{P})$ (see /Go/ for definitions of terms):

(1) $G(\mathcal{P})$ is regular
(2) the degree of a vertex is $m-1$
(3) $G(\mathcal{P})$ is connected
(4) $G(\mathcal{P})$ is a comparability graph
(5) $G(\mathcal{P})$ is a perfect graph
(6) $G(\mathcal{P})$ is planar iff $m \leq 4$.

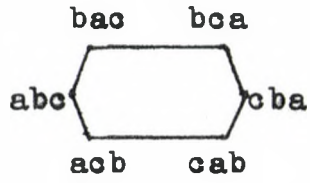


Figure 1.
Graph $G(\mathcal{P})$ for $M = \{a,b,c\}$.

As it was already mentioned in Introduction, we would like to consider the sorting process as a sequence of splits of sets of permutations. Since we proceed by comparisons, only sets of permutations which correspond to posets are admissible. For a given poset U we can find the corresponding set of permutations by extending U to linear orderings in all possible ways, and the collection of these linear extensions of U produces the set of all admissible permutations (see /Ri1/ for an excellent survey on linear extensions). The set of all linear extensions of a poset U will be denoted by E(U).

Our main "characterization" problem may be stayed as:

Let P be a set of permutations. Does there exist a poset U such that $E(U) = P$?

We shall show that the question can be answered investigating the graph G(P).

Definition 2. A nonempty set P of permutations is convex if for every $x,y \in P$, G(P) contains all shortest pathes from $G(\mathcal{P})$ with terminals x,y.

Using the example in Figure 1 it is clear that $P = \{abc, acb, cab\}$ is convex and $P = E(a > b)$ if we assume that the permutations are ordered in decreasing manner. On the other hand neither $\{abc, cab\}$ nor $\{abc, acb, cab, cba\}$ is convex and it is easy to see that in the both cases we are not able to find a proper poset. It can be shown that this fact holds generally:

Theorem 1 (Characterization). Let P be a set of permutations. There exists a poset U such that $E(U) = P$ iff P is convex.

Now we know that only convex sets of permutations are useful in a sorting process. But one more question remains:

How to picture a comparison ?

Let us again turn to the example in Figure 1. A comparison a:b yields two outcomes: $a > b$ or $a < b$. If $a > b$ then the corresponding set of permutations is $\{abc, acb, cab\}$ and if $a < b$ then it is $\{bac, bca, cba\}$. Both the sets of permutations are convex, i.e. the comparison a:b splits the original set of permutations into two convex subsets. It is clear that the graph $G(\mathcal{P})$ in Figure 1 can be splitted into two convex subgraphs just in three different ways and each of them corresponds to one of the possible comparisons a:b, b:c, a:c, respectively (Figure 2).

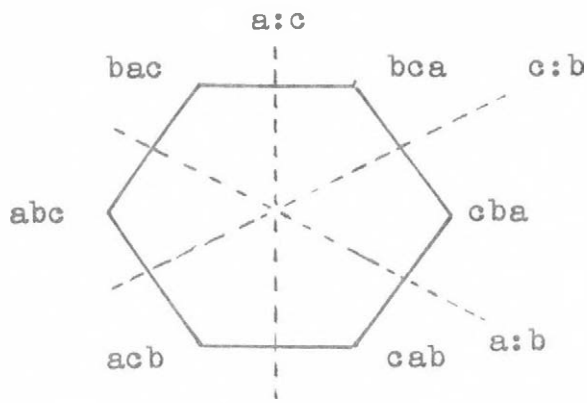


Figure 2. Splits caused by comparisons.

One can prove a generalization of the correspondence between a split of a graph and a comparison:

Theorem 2 (Splitting). (P_1, P_2) is a partition of a convex set P of permutations into two convex subsets P_1, P_2 iff there exist posets U, U_1, U_2 such that $P = E(U)$, $P_1 = E(U_1)$, $P_2 = E(U_2)$ and there exists a pair (a,b) such that

$$U_1 = (U \cup (a > b))^+ \quad \text{and} \quad U_2 = (U \cup (a < b))^+.$$

Moreover, for given partition the pair (a,b) is unique.

Now we can investigate comparison based processes without any care for comparisons; it suffices to ensure "convex" splits. This approach will be demonstrated in the following section.

3. Applications of the model

Here we present three different applications of the model described above.

(i) Optimal merging of 2 and n element sets.

We assume a problem of optimal merging of 2 and n element linearly ordered sets (disjoint) into a single linear order. The attribute "optimal" can be substituted either by "worst -case optimal" or by "average-case optimal"(here we assume that each possible output permutation is equally likely and only comparisons between merged elements are computationally relevant in the both cases) in the following.

At first we show how looks the graph of an input poset. We shall use an example with n=4. Let the merged sequences be $a_1 > a_2 > a_3 > a_4$ and $b_1 > b_2$. There are 5 possible positions of the b's among the a's as it is indicated in the Hasse diagram by numbers from 1 to 5 in Figure 3. Hence each possible

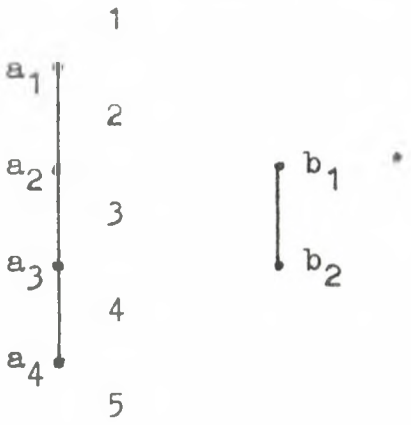


Figure 3.
Hasse diagram.

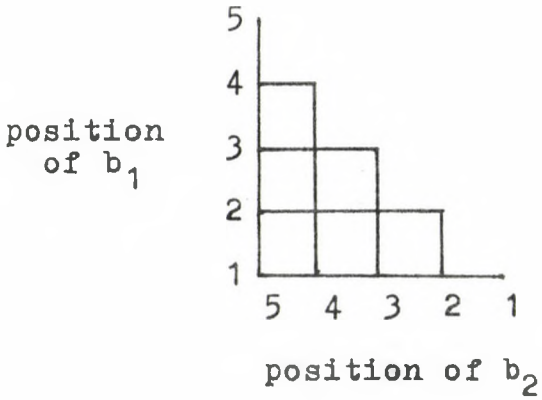


Figure 4.
Graph of permutations.

output permutation is uniquely determined by a tuple (i_1, i_2) where i_1 (i_2) is the final position of b_1 (b_2). Since $b_1 > b_2$, we have $i_1 \leq i_2$ and thus the corresponding graph of permutation has a nice grid structure (see Figure 4). We note this structure does not depend on the input range n.

It is easy to see that only horizontal and vertcal line cuts split the graph into two convex subgraphs (here we use "graph" instead "set of permutations") and therefore they correspond to some comparisons. After the first split (comparison) similar process is applied to the both subgraphs when we are constructing an algorithm. All such graphs can be arranged into a sequence, where a split of a graph with higher number in the sequence gives rise to two graphs with smaller numbers. Figure 5 shows the initial portion of the sequence of graphs.

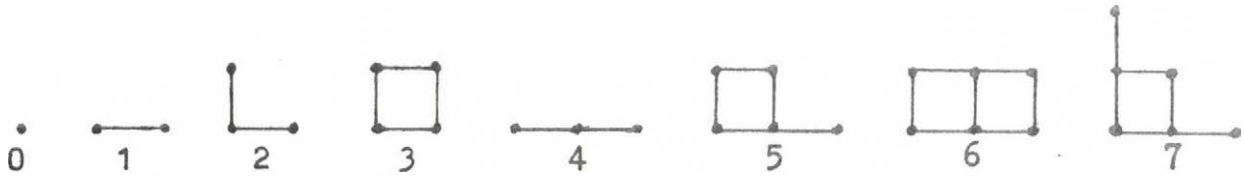


Figure 5. Sequence of graphs in 2:n merging.

The construction of optimal algorithms proceeds in this way: Let us assume that we have optimal sorting algorithms for graphs up to k-1 in the sequence (here we assume a general sorting algorithm - with an arbitrary poset as an input). Now try all possible convex cuts of the k-th graph and pick the best one (the optimal algorithms for graphs with smaller numbers are used to decide which cut gives the best algorithm) and it is the optimal algorithm for this graph.

It can be shown that the initial graph for 2:n merging is at the position $\frac{1}{6} n^3 + n^2 + \frac{5}{6} n$ in the sequence and there are maximally 2n possible cuts of this graph (and in all preceding graphs). Thus to find an optimal 2:n merging algorithm we have to check $O(n^3)$ graphs and $O(n)$ possible cuts in each of them. Therefore the complexity of this process is $O(n^4)$.

We note that the worst-case optimal 2:n merging algorithms are known (/HL/) while the average-case optimal algorithms are known only for some values of n (0,1,2,3,5,8,12,18,26,37,53,76, 108,154,... - see /HL/). Moreover, in all these average-case optimal algorithms the information theoretic lower bound is attained. Therefore we used a computer to find optimal algorithms for some small values of n.

Figure 6 shows the output - we computed the total (over all possible input permutations) number of comparisons and D(n) in the figure denotes the discrepancy of this total between the average-case optimal algorithm and the information theoretic lower bound (see /Kn/ for the information theoretic lower bound on average-case optimal merging).

n	D(n)	n	D(n)	n	D(n)	n	D(n)	n	D(n)	n	D(n)
0	0	1	0	2	0	3	0	4	1	5	0
6	1	7	1	8	0	9	2	10	5	11	1
12	0	13	2	14	7	15	7	16	2	17	0
18	0	19	4	20	11	21	22	22	15	23	7
24	2	25	0	26	0	27	4	28	11	29	22
30	35	31	35	32	21	33	11	34	5	35	1
36	0	37	0	38	1	39	9	40	20	41	35
42	53	43	74	44	88	45	69	46	51	47	35

Figure 6. D(n) in the optimal 2:n merging algorithms

Using the values of D(n) and denoting $N = \binom{n+2}{2}$ we can compute the average number of comparisons in the optimal 2:n merging algorithm as

$$\frac{D(n) + N.\lceil \lg_2 N \rceil + N - 2^{\lceil \lg_2 N \rceil}}{N}.$$

If D(n)=0, then for this n there exists an optimal merging algorithm which attains the information theoretic lower bound.

It is clear that similar algorithm (but with a greater complexity) can be also used to find optimal parallel 2:n merging algorithm.

(ii) Balancing poset extensions.

To obtain a good sorting algorithm we would like to perform splits of sets of permutations as balanced as possible. But what is the bound for such a balanced split? Kahn and Saks /KS/ had shown that an arbitrary convex set P of permutations with more than one element can be splitted into two convex subsets P_1, P_2 with

$$\frac{3}{8} < \frac{|P_1|}{|P_2|} < \frac{8}{3} .$$

However, we do not know a counterexample for the inequality

$$\frac{1}{2} \leq \frac{|P_1|}{|P_2|} \leq 2$$

(see the Unsolved problems section in the journal Order). A graph of permutations seems to be a good tool for proving the closer inequality. This will give a new information theoretic upper bound to a general sorting problem.

(iii) Parity problem.

This problem due to Chase (see p. 198 in /Kn/) may be stayed as: Let $a_1a_2...a_n$ be a permutation of $\{1,2,...,n\}$. Prove that any algorithm which decides whether this permutation is even or odd based solely on comparisons between the a's is equivalent to a sorting algorithm for sorting n element sets even though the "parity" algorithm has only two possible outcomes (i.e. the information theoretic lower bound gives one comparison).

Knuth /Kn/ presented a simple solution to this problem and Fredman /Fr/ shows the information theoretic lower bound is pitiful in much more cases.

Our graphs of permutations give a feasible argument to the equivalence between the parity and sorting problems. Since each convex set of permutations with more than one element contains

at least one pair of permutations with an opposite parity (neighboring permutations), only one element sets of permutations are allowed for the leaves of a decision tree for the parity problem. And this is just the same tree as that one for sorting n element sets.

4. Conclusion

The graphs of permutations bring a new insight not only into comparison based sorting like problems but into finite poset problems in general. We are sure that the presented list of problems where the graphs of permutations are useful is not exhaustive. And we recall, graphs of permutations are neither like potatoes nor like barrels /Ri2/, but "perfect" graphs!

References

/Fr/ Fredman, M.L.: How good is the information theory bound in sorting? Theoretical Computer Science 1(1976), pp. 355-361.

/Go/ Golumbic, M.C.: Algorithmic Graph Theory and Perfect Graphs, Academic Press, New York, 1980.

/HL/ Hwang, F.K., Lin, S.: Optimal merging of 2 elements with n elements. Acta Informatica 1(1971), pp. 145-158.

/Kn/ Knuth, D.E.: The Art of Computer Programming, Vol. 3 Sorting and Searching, Addison-Wesley, Reading, Mass., 1973.

/KS/ Kahn, J., Saks, M.: Balancing poset extensions. Order 1 (1984), pp. 113-126.

/Ri1/ Rival, I.: Linear extensions of finite ordered sets. In: Orders: Description and Roles, M.Pouzet, D.Richards eds., 1984, pp. 355-370.

/Ri2/ Rival, I.: The diagram. Order 2(1985), pp. 101-104.

Proc.IMYCS '86 October 13-17,1986
Smolenice Castle,ČSSR

THE BEHAVIOUR OF THE RATIO FUNCTION

Mária Kráľová

Mathematical Institute
Slovak Academy of Sciences
Obrancov mieru 49
814 73 Bratislava
Czechoslovakia

1. Introduction.

The aim of the present paper is to characterize the behaviour of the ratio function of a DOL system.

The first paper dealing with the analysis of the ratio function was the paper [5]. In that paper the behaviour of the ratio function of a DOL system is studied according to structural properties of the given DOL system.

The results from [5], concerning the ratio function determined by a monorecursive letter with the index of monorecursivity $t \geq 1$ and the ratio function determined by an expanding letter with the index of expansion $t=1$, are included in the part 3 of this paper. We give these results for the sake of the completness.

In the part 4 we continue the study of the properties of the ratio function for the case, when it is determined by an expanding letter with the index of expansion $t>1$. The characterization of such ratio function is given in theorem 4 (the main theorem of this paper).

2. Basic definitions.

In the following W always denotes a finite nonempty set

of symbols, W^* denotes the set of all words over W, including the empty word ε (the word with no symbols in it). If w is a word in W^*, then $\#_a w$ is the number of occurrences of the letter a in the word w.

The set of natural numbers $\{1,2,\ldots\}$ will be denoted by N, the set of nonnegative integers $\{0,1,2,\ldots\}$ by Z^+ and the set of nonnegative reals by $\mathcal{R}^+$.

Definition 1. A deterministic Lindenmayer system without interactions (abbreviated DOL system) is an ordered triple $G=(W,h,w)$, where W is a finite nonempty set of symbols, h is a homomorphism from W into W^* and $w \in W^*$ is an initial word (called axiom).

Definition 2. For any DOL system $G=(W,h,w)$ a letter $a \in W$ is called

mortal ($a \in M$) if $h^i(a)=\varepsilon$ for some $i \in Z^+$,

recursive ($a \in R$) if $h^i(a)=v_1 a v_2$ for some $i \in N$, $v_1, v_2 \in W^*$,

monorecursive ($a \in MR$) if $h^i(a)=v_1 a v_2$ for some $i \in N$, $v_1, v_2 \in M^*$,

expanding ($a \in E$) if $h^i(a)=v_1 a v_2 a v_3$ for some $i \in N$, $v_1, v_2, v_3 \in W^*$,

b-mortal ($a \in b\text{-}M$) for $b \in W$ if there exists a number $i_0 \in N$, for which $\#_b h^i(a)=0$, $i \geq i_0$,

accessible from a letter $b \in W$ ($a \in U(b)$) if there is a number $i \in N$ such that $\#_a h^i(b) \neq 0$.

Definition 3. Let $G=(W,h,w)$ be a DOL system and $a \in W$. An equivalence class

$$[a]_G = \{b \in W;\ b \in U(a) \text{ and } a \in U(b)\}$$

is called the level of the DOL system G generated by a.

Remark. The subscript G will be ommited in the notation always when it is clear which G is considered.

Definition 4. The level [a] is said to be mortal, recursive, monorecursive, expanding, b-mortal if the letter a is mortal, recursive, monorecursive, expanding, b-mortal, respectively.

Definition 5. Let G=(W,h,w) be a DOL system. A letter $a \in W$ is called mortal with the index of mortality t ($a \in M^{(t)}$), recursive with the index of recursivity t ($a \in R^{(t)}$), monorecursive with the index of monorecursivity t ($a \in MR^{(t)}$), expanding with the index of expansion t ($a \in E^{(t)}$), if t is the smallest number, for which the condition of mortality, recursivity, monorecursivity, expansion, respectively, is satisfied.

Definition 6. Let G=(W,h,w) be a DOL system, let $w'=x_1x_2\ldots x_n$, where $x_i \in W$ for i=1,...n, and let $w=y_1x_1x_2\ldots x_ny_2$, where $y_1,y_2 \in W^*$. Then w' is called the subaxiom of the axiom w.

Definition 7. An ordered 4-tuple G'=(W,h,w,w') is called the DOL system with a subaxiom, if (W,h,w) is a DOL system and w' is a subaxiom of the axiom w.

Definition 8. Let G'=(W,h,w,w') be a DOL system with a subaxiom and let $a \in W$. The function $r_a: Z^+ \to \mathcal{R}^+$, defined as follows

$$r_a(i) = \begin{cases} (\sharp_a h^i(w'))(\sharp_a h^i(w))^{-1}, & \text{if } \sharp_a h^i(w) \neq 0 \\ \text{is not defined, if } \sharp_a h^i(w) = 0, \end{cases}$$

is called the ratio function of G' determined by a.

3. Ratio functions with the constant behaviour.

Consider a DOL system with the subaxiom G'=(W,h,w,w'). After what conditions has the ratio function of G' the constant

behaviour ? In [5] the reader can find the answer to this question in the following form:

Theorem 1. Let G' be a DOL system with a subaxiom, let $[a]$ be its monorecursive level with the index of monorecursivity 1 and let there hold for each of the letters $b \in W$, $b \neq a$ one of the following conditions:

1. $b \in a\text{-}M$,
2. $b \notin R$ and $a \in U(b)$.

Then there exists a natural number i_0 such that

$$r_a(i) = pq^{-1}$$

for all $i \geq i_0$, the nonnegative integer p and the natural q.

Sketch of the proof.

The assumptions $b \neq a$, $a \in MR^{(1)}$, $b \in a\text{-}M$ imply $\sharp_a h^i(b) = 0$ for all $i \in N$.

Denote

$$A = \{b \in W;\ b \notin R \text{ and } a \in U(b)\}$$

and assume

$$\text{card } A = s.$$

Then we can write

$$A = \{b_1, b_2, \ldots b_s\}.$$

It is clear that for every $i > s$ and $b_k \in A$ there holds

$$\sharp_a h^{i+1}(b_k) = \sharp_a h^i(b_k).$$

It suffices to put $i_0 = s$ and the assertion of the theorem is proved for

$$p = \sharp_a w' + \sum_{b_k \in w'} \sharp_{b_k} w' \sharp_a h^{i_0}(b_k),$$

$$q = \sharp_a w + \sum_{k=1}^{s} \sharp_{b_k} w \, \sharp_a h^{i_0}(b_k), \quad k = 1, 2, \ldots s.$$

Theorem 2. Let G' be a DOL system with a subaxiom and let $a \in W$, $a \in E^{(1)}$. If for any $b \in W$, $b \neq a$ it holds either $a \notin U(b)$

or $a \in U(b)$ and $b \notin R$,
then there is a nonnegative integer number i_0 such that

$$r_a(i) = pq^{-1}$$

for all $i \geq i_0$, the nonnegative integer p and the natural number q.

We outline the proof of this theorem.

Consider $a \in E^{(1)}$, $b \neq a$ such that $a \notin U(b)$. Then put

$$\sharp_a w = p, \quad \sharp_a w' = q$$

and

$$\sharp_a h(a) = m.$$

Hence

$$\sharp_a h^i(a) = m^i \quad \text{for } i \in Z^+.$$

So, in this case $i_0=0$.

Assume that $a \in U(b)$ and $b \notin R$. The nonrecursivity of the letter b implies $b \notin U(a)$.

Let $b_1,\ldots b_s$ be all letters of the axiom w such that $b_i \neq a$, $a \in U(b_i)$ and $b_i \notin R$ for $i=1,\ldots s$. Put $i_0=s$. Then it holds

$$r_a(i) = r_a(i_0) = (m^{i_0} \sharp_a w' + \sum_{b_i \in w'} \sharp_a h^{i_0}(b_i) \sharp_{b_i} w')(m^{i_0} \sharp_a w +$$

$$+ \sum_{i=1}^{s} \sharp_a h^{i_0}(b_i) \sharp_{b_i} w)^{-1}, \quad i \geq i_0$$

4. Ratio functions with the periodic behaviour.

In the previous part we have considered the ratio function determined by a monorecursive letter with the index of monorecursivity 1 (theorem 1) and the ratio function determined by an expanding letter with the index of expansion 1 (theorem 2). What happens, if we shall consider the ratio function determined by a monorecursive, respectively expanding, letter with the index of monorecursivity, resp. expansion, greater than 1? The next theorems indicate that the ratio functions are perio-

dic. In this sense the theorems 3, 4 are extensions of previous ones.

Theorem 3. Let $G'=(W,h,w,w')$ be a DOL system with a sub-axiom, let $a\in W$, $a\in MR^{(t)}$, $t>1$, $t\in N$, let for any $b\in W$, $b\neq a$ one of the following conditions holds: 1. $a\notin U(b)$,

2. $a\in U(b)$, $b\in MR$ or $b\notin R$.

Then there exists a number i_0 that for all $i\geq i_0$, $i\in N$ the ratio function $r_a(i)$ is periodic with the period t.

Sketch of the proof.

For $b\in W$ such that $a\notin U(b)$ we have $\sharp_a h^i(b)=0$ for $i\in Z^+$. The property $a\in MR^{(t)}$, $t>1$, implies: there is a finite sequence $b_1,\ldots b_{t-1}\in W$, for which $\sharp_a h^{nt-k}(b_k)\neq 0$ for $k=1,\ldots t-1$, $n\in N$. Suppose that c_s, $s=1,\ldots m$, are all letters of the alphabet W satisfying the conditions $a\in U(c_s)$, $c_s\notin R$. Then there exists the smallest number i_s, for which

$$h^{i_s}(c_s)\in W^*aW^*.$$

Denote

$$C=\{s;\ s=1,\ldots m,\ i_s(\mathrm{mod}\ t)=0\}\ \text{and}\ i_0=\max_{s\in C} i_s + t.$$

Let the axiom be given by $w=x_1\ldots x_r$, $r\in N$. Then

$$r_a(i_0+nt)=(\sharp_a w'+\sum_{x_j\in B'}\sharp_a h^{i_0}(x_j))(\sharp_a w+\sum_{x_j\in B}\sharp_a h^{i_0}(x_j))^{-1},$$

where $n\in N$, $B=\{x_j;\ j=1,\ldots r,\ x_j=c_s,\ s\in C\}$, $B'=\{x_j\in B, \sharp_{x_j} w'\neq 0\}$. We proved that

$$r_a(i_0+nt)=r_a(i_0),\quad n\in N.$$

In the same way it is possible to prove that

$$r_a(i_0+nt+l)=p_l q_l^{-1}\quad \text{for any } n\in N \text{ and } l=1,\ldots t-1.$$

We shall say that a letter $b\in W$ satisfies the condition (C), if the production rule for b has the form $h(b)=v$, where $v\in(a-M\cup d)^*$, $d\in[a]$, i.e. the word v does not contain two different letters occurring in the level $[a]$.

This condition guarantees that the letters of the same expanding level have the same index of expansion.

Proposition 1. Let $G=(W,h,w)$ be a DOL system, let $a \in W$, $a \in E^{(t)}$, $t \geq 1$, let any $b \in W$, $b \in [a]$ satisfies the condition (C). Then the level $[a]$ contains exactly t letters.

The assertion of the following proposition together with that one above will be used in the proof of next theorem.

Proposition 2. Let $G=(W,h,w)$ be a DOL system, let $a \in W$, $a \in E^{(t)}$, $t > 1$. If any $b \in W$, $b \in [a]$ satisfies the condition (C) then each letter occurring in the level $[a]$ is expanding with the index of expansion exactly t.

The proofs of both propositions can be found in [6].

Theorem 4. Let G' be a DOL system with a subaxiom. Let $a \in E^{(t)}$, $t > 1$, $a \in W$ satisfying the condition (C). Let for any $b \in W$, $b \neq a$ one of the following conditions holds

1. $a \notin U(b)$,
2. $a \in U(b)$ and $b \notin R$,
3. $b \in [a]$ and b satisfies the condition (C).

Then there exists a nonnegative integer i_0 such that for any $i \geq i_0$, $i \in N$, the ratio function r_a is periodic with the period t.

Sketch of the proof.

Let us denote $\sharp_a h^t(a)=m$ and b_k, $k=1,\ldots r$, all of the letters of the alphabet W, for which the assumption 2. is satisfied. Then for every b_k there is the smallest number i_k that

$$h^{i_k}(b_k) \in W^* a W^*, \quad k=1,\ldots r.$$

Let

$$T = \{k;\ i_k \pmod t = 0,\ k=1,\ldots r\} \text{ and } i_0 = \max_{k=1,\ldots r} i_k + t.$$

The definition of the number i_0 implies that there is a number $n_0 \in N$ with the property

$$i_0 = n_0 t.$$

According to the proposition 1 the level $[a]$ consists of t letters. Let us suppose that $[a]=\{a,c_1,\ldots c_{t-1}\}$ and

$$\sharp_{c_j} h^j(a) \neq 0, \quad j=1,\ldots t-1.$$

If we denote $T' = \{k \in T;\ \#_{b_k} w' \neq 0\}$, we can express the ratio function determined by a as

$$r_a(i_0) = (m^{n_0}\#_a w' + \sum_{k \in T'} \#_a h^{i_0}(b_k)\#_{b_k} w')(m^{n_0}\#_a w + \sum_{k \in T} \#_a h^{i_0}(b_k)\#_{b_k} w)^{-1} = r_a(i_0+nt),\ n \in Z^+.$$

It remains to prove only that the relation

$$r_a(i_0+nt+l) = r_a(i_0+l)$$

is true for $l=1,\ldots t-1$. Then it suffices to put

$$T_l = \{k;\ i_k \pmod t = l,\ k=1,\ldots r\}$$

and

$$T_l' = \{k \in T_l;\ \#_{b_k} w' \neq 0\}.$$

REFERENCES

[1] Herman G.T., Vitányi P.M.B.: Growth functions associated with biological development, American Math. Monthly 83, 1976, 1-15

[2] Hromkovič J., Kelemenová A.: On kinetic models of cell population, Proc. of The 3-rd Int. Symp. of System Simulation in Biology and Medicine, Prague, 1982, Microfishe No 735

[3] Kelemenová A.: Levels in L-systems, Math. Slovaca 33, 1983, 1, 87-97

[4] Kelemenová A.: Complexity of L-systems, Chapter in book G. Rozenberg, A. Salomaa (eds.): The book of L, Springer Verlag, Berlin-Heidelberg, 1986, 179-191

[5] Kráľová M.: Constant ratio function of Lindenmayer systems, Math. Slovaca 35, 1985, 3, 283-294

[6] Kráľová M.: Ratio functions determined by an expanding letter, submited to Kybernetika

[7] Lindenmayer A., Rozenberg G.: Automata, Languages, Development, North Holland, Amsterdam 1976

[8] Rozenberg G., Salomaa A. (eds.): The Mathematical Theory of L-systems, Academic Press, New York, 1980

Proc. IMYCS '86 October 13-17, 1986
Smolenice Castle, ČSSR

THE SYMMETRIC DIFFERENCE PROBLEMS ON GRAPHS

Mirko Křivánek

Research Institute of Mathematical Machines

Loretánské nám. 3, 118 55 Praha 1

Czechoslovakia

A b s t r a c t. The aim of this contribution is to draw the attention to the systematization of the NP-complete results on graph problems. Within the context of [1,8] the discussion is relative to the symmetric difference problems on graphs.

I. INTRODUCTION

A large variety of computational problems defined on graphs are known to be NP-complete, and hence there are no polynomial algorithms solving them exactly. Throughout the last decade the list of NP-complete problems has expanded steadily. Recently efforts have begun towards systematizing NP-completeness proofs and attacking groups of similar problems. Some systematic approaches have been achieved to problems defined on graphs. In particular, more recently, many researchers considered so-called "edge-deletion" or "edge-contraction" problems on graphs, cf. [1,8] . These problems under consideration are generally formulated as follows :

OP($\mathcal{P}$) : INSTANCE : A graph G with the set of vertices $V = \{v_1, \ldots, v_n\}$, a class $\mathcal{G}_{\mathcal{P}}$ of graphs on V having the property $\mathcal{P}$, positive integer k ;

QUESTION : Does it holds that

$$OP(G, \mathcal{G}_{\mathcal{P}}) \leq k \ ? \ ,$$

where

$OP(G, \mathcal{G}_{\mathcal{P}})$ denotes the minimum number of edge-operations, such as deletion, addition, contraction ... ,that are needed to perform on a graph G in order to obtain a graph G´ from $\mathcal{G}_{\mathcal{P}}$.

In what follows we shall investigate the NP-completeness of $OP(\mathcal{P})$ for some graph properties $\mathcal{P}$ supposing that OP stands for the operation addition and/or deletion of an edge. It is easy to see that in this case

$OP(G, \mathcal{G}_{\mathcal{P}}) = k$ $\Leftrightarrow$ there is a graph $G' \in \mathcal{G}_{\mathcal{P}}$ on the same vertex-set as G such that $|G \triangle G'| = k$, where

$\triangle(G,G') = |G \triangle G'| = |(E(G) - E(G')) \cup (E(G') - E(G))|$ is the cardinality of the symmetric difference of graphs G and G´.

Now, our discussion will be parallel to problems where "OP" concerns only deletion of an edge. We shall use this notation :

$OP(\mathcal{P}) \equiv \triangle(\mathcal{P})$ ..... symmetric difference problems,

$-(\mathcal{P})$ ..... edge-deletion problems

II. RESULTS

Throughout this paper let $V = \{v_1, \ldots, v_n\}$ be a finite set. Further let $\mathcal{G}_c$ be a class of all clique graphs on V, i.e. graphs whose all components are complete graphs. Referring to [4,p.68] the following theorem is straightforward :

T h e o r e m 1

Problem $-(G, \mathcal{G}_c)$ is NP-complete even when it is restricted to graphs of maximum degree 6 and without subgraphs isomorphic to K_4.

Using different proof technique the analogous result has been obtained for problem $\Delta(G, \mathcal{G}_c)$, cf. [5]:

T h e o r e m 2
Problem $\Delta(G, \mathcal{G}_c)$ is NP-complete even when restricted to graphs with maximum degree 6 and without K_4.

Comparing proofs of Theorem 1 and Theorem 2 one can see that essentially problems $-(G, \mathcal{G}_c)$ and $\Delta(G, \mathcal{G}_c)$ are not the same as one could expect by examining the problems $-(\mathcal{B})$ and $\Delta(\mathcal{B})$, where $\mathcal{B}$ stands for denoting the bipartiteness of graphs. Indeed, it is senseless to add edges to a graph to make it bipartite. On the other hand the problem $-(\mathcal{B})$ is known to be NP-complete [4,p.196] and consequently the problem $\Delta(\mathcal{B})$ is also NP-complete.

Now, another example of the same situation will be given. Let $\mathcal{G}_R$ denote the class of all Robinson graphs. Recall that a graph is said to be Robinson if there is a permutation ς such that when it is simultaneously applied to rows and columns of a graph incidence matrix the following conditions are satisfied :

(i) $a_{\varsigma(i),\,\varsigma(i)+1} \leq a_{\varsigma(i),\,\varsigma(i)+2} \leq \ldots \leq a_{\varsigma(i),n}$

(ii) $a_{\varsigma(i),\,\varsigma(i)+1} \leq a_{\varsigma(i)-1,\,\varsigma(i)+1} \leq \ldots \leq a_{1,\,\varsigma(i)+1}$,

$i = 1,\ldots,n$.

In [6] the following theorem was proved :

T h e o r e m 3
Problem $\Delta(G, \mathcal{G}_R)$ is NP-complete even when it is restricted to planar graphs which are cubic and such that each face has at least 5 edges.

Examining the proof of Theorem 3 we can see that the problem $\Delta(G, \mathcal{G}_R)$ is again the same problem as the problem $-(G, \mathcal{G}_R)$ but only with respect to the restricted instances as it is described in Theorem 3. Of course, for prooving the NP-completeness of $-(G, \mathcal{G}_R)$ it is sufficient.

In further discussion we shall present some new results concerning symmetric difference problems on graphs. We shall

introduce and investigate the symmetric difference problem for two graphs $G,H \in \mathcal{G}_p$ as the problem of computation

$$\Delta(G,H) = \min \{ \Delta(G,H') ; H' \cong H \} .$$

Therefore we are interested in determining the minimum number of edge deletions/additions needed to perform in G in order to obtain a graph isomorphic to H.

Our main result is read as follows :

T h e o r e m 4

Let T_1, T_2 be two trees of the same order , k be positive integer. Then it is NP-complete to decide whether $\Delta(T_1,T_2) \leq k$.

Proof.

Since the test for isomorphism of two trees can be done in polynomial time (e.g. [3]) the aforementioned problem is trivially in the class NP. We shall give a polynomial transformation from the known NP-complete problem ERD : INSTANCE : Two trees T_1,T_2 of the same order, positive integer k ;

QUESTION : Is $erd(T_1,T_2) \leq k$?

By $erd(T_1,T_2)$ we mean the edge-rotation distance between T_1 and T_2 , i.e. the minimum number of edge-rotations needed to transform T_1 into a tree isomorphic to T_2 .

(T' arises from T by one edge-rotation $\Leftrightarrow$ $erd(T,T') = 1$ $\Leftrightarrow$ $T' \cong T - \{u,v\} + \{u,w\}$ where $\{u,v\} \in E(T)$, $\{u,w\} \notin E(T)$

The NP-completeness of ERD was established in [7] .

Let T_1, T_2 be two trees on the same set of vertices V such that $\Delta(T_1,T_2) = k$. Note that k is even. Let us assume the graph $G = (V,E) = T_1 \Delta T_2$. Further let us denote $V_1 = E \cap E(T_1)$, $V_2 = E \cap E(T_2)$. Let us suppose that the graph $T_1 \Delta T_2$ is polynomially computable.

Let us define the bipartite graph $G_b = (V_1,V_2,E_b)$ such that $\{ e_1,e_2 \} \in E_b \Leftrightarrow e_1 \cap e_2 \neq \emptyset$. Let M be the maximum matching in G_b. Note that V_1 corresponds to edges that were deleted from T_1 while V_2 represents edges added to T_1 in order to obtain from T_1 a graph isomorphic to T_2.

As both T_1 and T_2 are acyclic graphs where any addition of an edge forms a cycle it holds that for any T', $\Delta(T,T')=2$

$$erd(T_1,T') = 1 \iff T' \cong T_1 - e_1 + e_2 \,,\; e_1 \cap e_2 \neq \emptyset$$
$$= 2 \iff T' \cong T_1 - e_1 + e_2 \,,\; e_1 \cap e_2 = \emptyset$$

Altogether it yields that $e_1, e_2 \in T_1 \Delta T'$

$$\Delta(T_1,T_2) = k \iff erd(T_1,T_2) = k - |M|.$$

(the proof is visualised in Figure 1).

Thus given an input instance (T_1,T_2,k) of ERD using a polynomial algorithm for determining $\Delta(T_1,T_2)$ we can decide in polynomial time whether $erd(T_1,T_2) \leq k$, a contradiction.

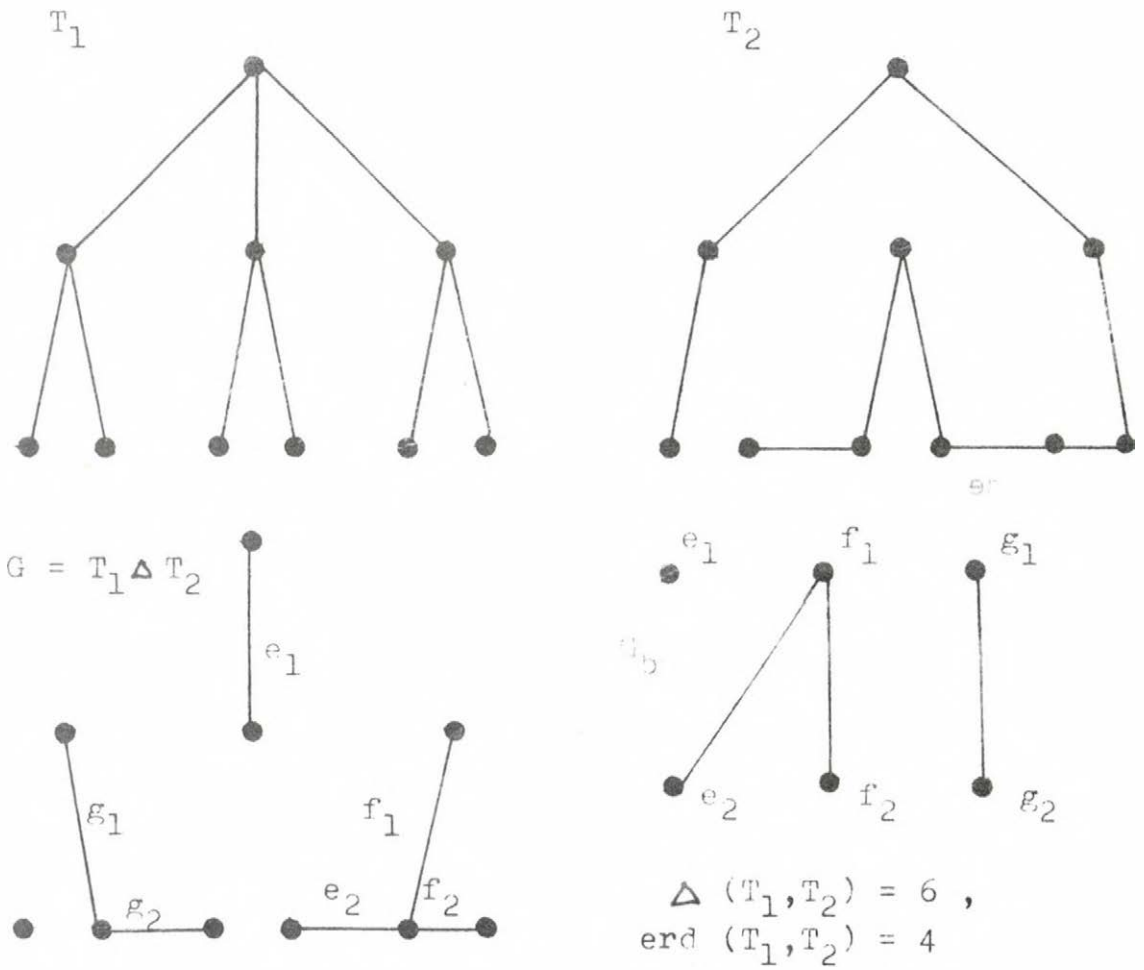


Figure 1.

Let $\mathcal{G}_{\mathcal{T}}$ be the class of all trees on V. We obtained the corollary :

T h e o r e m 5
The problem $\Delta(\mathcal{T})$ is NP-complete even when restricted to the class $\mathcal{G}_{\mathcal{T}}$.

On the other hand we have

T h e o r e m 6
The problem $-(\mathcal{T})$ is polynomially solvable .

Proof.

We are only to decide for two trees if they are isomorphic or not. It can be done in polynomial time [3] .

To close our discussion it remains to find an example of a graph property $\mathcal{P}$ such that $\Delta(\mathcal{P})$ is polynomially solvable while $-(\mathcal{P})$ has no polynomial solution.

Let $\mathcal{G}_{\mathcal{S}}$ denote the class of all split graphs, i.e. graphs which are union of exactly one complete and one discrete graph. Taking into account the known NP-complete problem "CLIQUE", see [4,p.194] , the following theorem is selfevident :

T h e o r e m 7
The problem $-(G, \mathcal{G}_{\mathcal{S}})$ is NP-complete.

On the other hand in [2] the polynomial algorithm for $\Delta(G, \mathcal{G}_{\mathcal{S}})$ has been developed .

III. CONCLUSION

Let P denote the class of all polynomially solvable problems and NPC the class of all NP-complete problems. In this paper we have introduced the symmetric difference problems on graphs. Their NP-completeness was discussed parallel with NP-completeness of corresponding edge-deletion problems. Unfortunately there is no mechanism for dealing with symmetric difference problems and edge-deletion problems altogether. In the subsequent table all possibilities of the NP-completeness "behaviour" are summarized.

The reader sholud have little difficulties to find an example of the pair of "P"-symmetric difference problem and "P"-edge-deletion problem for some graph property .

$\mathcal{P}$:	$\mathcal{C}$	$\mathcal{B}$	$\mathcal{R}$	$\mathcal{T}$	$\mathcal{S}$
$\triangle(\mathcal{P})$	NPC	NPC	NPC	NPC	P
$-(\mathcal{P})$	NPC	NPC	NPC	P	NPC

R e f e r e n c e s :

[1] T.Assano, T.Hirata : Edge-deletion and Edge-contraction problems. Proc.1982 ACM STOC conf.,245-254.

[2] P.L.Hammer, B.Simeone : The Splittance of a Graph. Res.Rep.CORR 77-39, Faculty of Math., Univ.Waterloo,1977

[3] J.E.Hopcroft,J.K.Wong : Linear Time Algorithm for Isomorphism of Planar Graphs. Proc 1974 ACM STOC conf.

[4] M.R.Garey,D.S.Johnson : Computers and Intractability : a Guide to the Theory of NP-completeness.Freeman,1979.

[5] M.Křivánek,J.Morávek : NP-Hard Problems in Hierarchical Tree Clustering.(to appear in Acta Informatica,1986)

[6] M.Křivánek : A Note of the Computational Complexity of Hierarchical Overlapping Clustering. Aplikace matematiky 30(1985),453-460.

[7] M.Křivánek : A Note on the Computational Complexity of Computing the Edge-rotation Distance in Graphs, 1985 (submitted)

[8] M.Yannakakis : Edge-deletion Problems. SIAM J.Comput., 10(1981),297-309.

Proc. IMYCS '86 October 13-17, 1986
Smolenice Castle, ČSSR

MULTI-LAYER CHANNEL ROUTING

Erika Kupková

Department of Theoretical Cybernetics
Comenius University
842 15 Bratislava
Czechoslovakia

Introduction

One of the most difficult and important problems in VLSI theory is the problem of placement and interconnection: given the set of cells with terminals on its boundaries, find a placement of cells and route the wires between the corresponding terminals so that the whole layout takes a minimal area. This problem was shown to be NP-complete. One way to solve this problem is to partition it into several simpler subproblems. One of such subproblems is the channel routing problem (CRP).

In this paper we shall deal with a multi-layer channel routing, where wires can be routed in more layers. The model we shall use is the generalization of the well-known two-layer Vertical-Horizontal model (VH). In the VH model, one layer serves for routing the vertical and the second for routing the horizontal segments of wires. In our model of 2k+1 layer channel, vertical and horizontal layers alternate beginning with vertical one. We shall call this model 2k+1 layer VH model and use the notion $(VH)_kV$ for it.

In this paper we present one provably good algorithm for routing 2-terminal nets, which solves a CRP of density d in 2k+1 layer channel in $\lceil d/k \rceil + 1$ tracks without doglegging. Then we shall discuss some possible generalizations

of this algorithm for multi-terminal nets.

Basic definitions and concepts

We shall briefly review some basic notions here. The reader should refer to [2,4] for more details and remaining unexplained notions.

Let us consider a rectangle channel placed in a grid consisting of rows (tracks) and columns. Numbered pins (terminals) are placed in the top and bottom rows of the channel. Pins with the same number constitute a net which is to be electrically connected. The CRP (Channel Routing Problem) is fully defined by a list of top and a list of bottom terminals with "0" (a space) inserted appropiately to indicate that in the given column there is no terminal. An instance of a CRP is shown in Fig. 1a. Fig. 1b illustrates another way of presenting the same CRP - each net is represented by a line segment on which the lower and upper terminals of the net are indicated.

number of column	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
list of upper t.	4	0	1	0	5	0	6	0	2	7	0	0	7	0	8	3	0	9	9	4	0
list of lower t.	0	5	0	0	6	1	7	1	0	2	4	2	8	2	9	9	0	0	0	0	3

Figure 1a

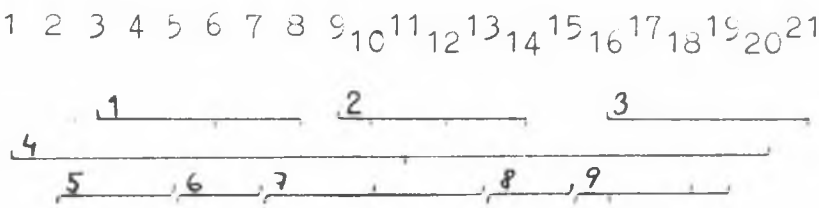


Figure 1b

To solve the CRP means to design connections between the terminals of the net in such a way that each terminal of the net is electrically connected and no two terminals from different nets are electrically connected. In the case when

each net has exactly 2 terminals we shall speak about 2-terminal CRP.

The solution of the CRP from Fig.1 (see Fig.2) is constucted in the two-layer VH model. In places, where two layers are electrically connected, there are contacts (circles). In this solution each net has only one horizontal segment. Routing models, which do not allow the use of several tracks per net are called models without doglegging. In Fig. 3 we can see a solution of the problem from Fig.1 without doglegging in three-layer VHV model.

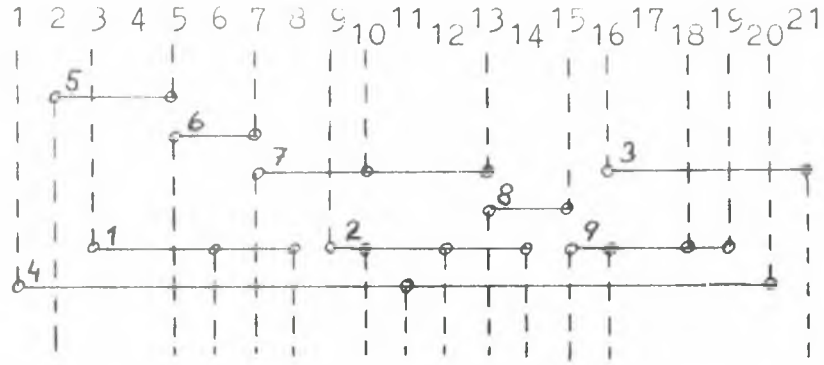


Figure 2

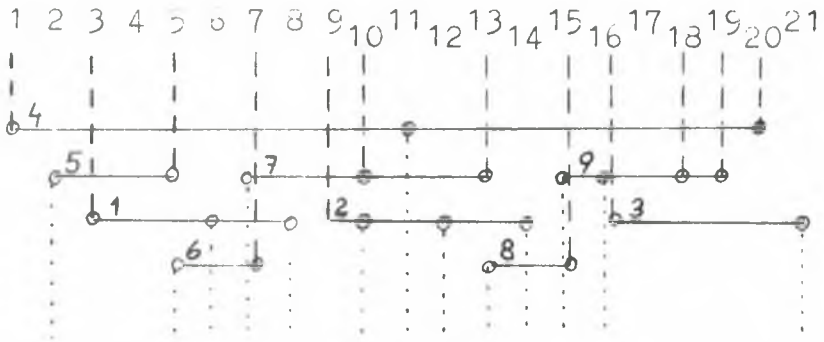


Figure 3

layer 1 – – – layer 2 ——— layer 3 · · · · ·

We can see the number of tracks in the first solution is more then in the second. The reason is the following: in the two-layer VH model there are vertical constraints - net 5 has to lie above net 6, net 6 has to lie above net 7, and so on, otherwise there would be a conflict in the vertical layer. In the VHV model we have 2 layers for routing verti-

cal segments, therefore there are no vertical constraints.

By [2] the number of tracks needed to solve the CRP in VHV model equals exactly to the density of this problem. Density of the CRP in column i is the number of nets intersecting this column (each of them has terminals on both sides of the column i) plus the number of nets starting or ending in this column (they have terminals in the column i and on one side of this column). Density of a CRP is the maximum over densities in all columns.

The number of tracks in two-layer VH model without doglegging cannot be smaller then the density of the problem but it can be much greater (depending on the number of vertical constraints).

By developing various VLSI manufacturing technologies, routing in more layers becomes possible. In [1] we can see one approach to solving the CRP in multi-layer channel. In this approach a channel model is used where no layer is strictly horizontal or vertical. The number of tracks needed in the presented algorithm equals to $\lceil d/m \rceil + 1$, where d is the density of the CRP, L is the number of layers and $m \leq L/2 - 2$ is the number of wires on different layers which are allowed to run on top of each other.

In this paper we introduce another model of multi-layer channel - the Vertical-Horizontal model. The $(VH)_k V$ model of the channel is the 2k+1 layer channel where vertical and horizontal layers alternate, beginning with a vertical one. So, in $(VH)_k V$ model of the channel there are k horizontal and k+1 vertical layers.

Results

First, we give a description of an algorithm for an optimal solution of the 2-terminal CRP in the $(VH)_k V$ model.

The main idea of the algorithm is the following: the wire for each nontrivial net will consist of 3 segments. One horizontal segment starting in the column of the left termi-

nal of the net and ending in the column of the right terminal of the net. Further, two vertical segments which will connect the horizontal segment and the terminals. The whole net will occupy two or three neighbouring layers. The net with both terminals in the same column (trivial net) has only one vertical segment which will occupy one (whichever) vertical layer.

In 2k+1 layer channel we consider each track to consist from k subtracks (1 subtrack for each horizontal layer). We distribute the horizontal segments of nets of the CRP among subtracks according to the following conditions:

1. segments on subtracks cannot overlap
2. no two nets having terminals in the same column are assigned to subtracks of the same track.

We can do this distribution using at most $\lceil d/k \rceil + 1$ tracks in the following way: We assume we have a list of nets ordered according to the left terminals of the nets. In each step, we take the next net (let i is the number of the column where this net starts) from the list and assigne it to the first free subtrack (a subtrack which does not contain any net starting, ending or intersecting column i) of a track which does not contain any net having terminal in column i. Since among $\lceil d/k \rceil + 1$ tracks there are always at least k free subtracks (the density of the CRP is d and the number of subtracks is $\geq d+k$) such a track always exists.

When the distribution of nets among subtracks (and thus among horizontal layers) is done we have to assign the layers for vertical segments. We shall use the following procedure:

If there is a net having both terminals in the same column we simply connect these terminals in any vertical layer. If two nets are assigned to the same horizontal layer l and their vertical segments overlap then one of these segments we route in vertical layer l and the second in vertical layer l+1. All remaining vertical segments of nets assigned to horizontal layer l we route in vertical layer l.

We note that in no vertical layer vertical segments

overlap. The reason is the following:

In each column there are at most 2 terminals, thus in each column there are at most 2 vertical segments. Let a and b are nets having vertical segments in the same column. In the case when a and b are assigned to different horizontal layers l and j, these vertical segments are routed in different vertical layers l and j. In the case when a and b are assigned to the same horizontal layer l and their vertical segments overlap, one of these segments is routed in vertical layer l and the second in vertical layer l+1. Therefore, vertical segments do not overlap in any vertical layer.

In places where horizontal and vertical segments of a net intersect, we place the contact between the corresponding layers.

In figure 4 we can see an example of the CRP of density 8 routed in 5 layers.

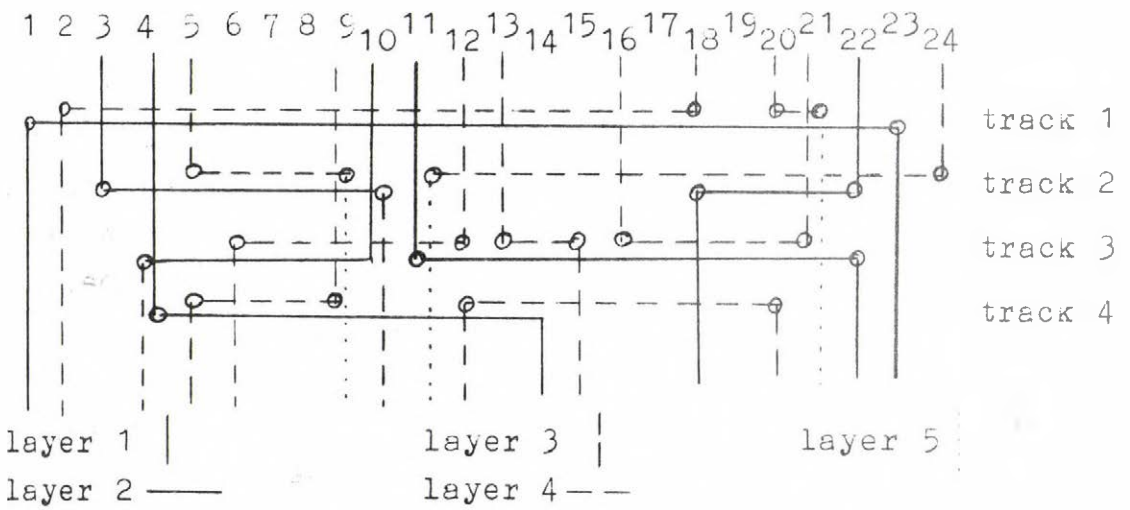


Figure 4

Let us note that using at most k horizontal layers to solve a CRP of density d we need at least $\lceil d/k \rceil$ tracks, so our algorithm really gives a good result.

Clearly, the algorithm works in time $O(n)$, where n is the number of nets. We note there was an assumption the list of nets be sorted. If no, we need additional $O(n \log n)$ time

to sort the nets.

In the case of multi-terminal nets the situation is a little bit more complicated, unless we do not allow two contacts in one gridpoint (for example one between layers 1 and 2 and the second between layers 6 and 7). In such case we can use the same algorithm as for 2-terminal nets. The only difference is that during the distribution of nets among subtracks we forget all middle terminals (those terminals of a net which are neither the leftmost nor the rightmost terminal) and consider the multiterminal net to be a 2-terminal net. Following the distribution among subtracks we consider all terminals of the nets and route the vertical segments.

In case we do not allow two contacts in one gridpoint there are instances of multiterminal CRP having no good solution without doglegging. Let us take for example an instance of CRP where for each two nets there is a column in which both nets have a terminal. In this case no two nets can be at the same track (otherwise there must be two contacts in one gridpoint on this track) and so we could not use the advantage of more horizontal layers.

Therefore in some cases we have to use doglegging, which means that we route one net in more tracks. Because of the lack of space we do not present here an algorithm which solves the multiterminal net problem in the model where 2 contacts in one gridpoint are not allowed, we only note that it can be solved in $\lceil d/k \rceil + 2$ tracks.

Acknowledgement

I would like to thank Branislav Rovan for his comments concerning this work.

Refferences

1 S.E.Hambrusch (1985): Channel routing algorithms for

overlap models, IEEE Trans. CAD, vol. CAD-4, No1, pp. 23-30

2 R.Srinivasan, L.M.Patnaik (1984): Two algorithms for three-layer channel routing, Computer aided design, vol. 16, No5, pp. 264-271

3 J.D.Ullman (1984): Computational aspects of VLSI, Computer Science Press

4 T.Yoshimura, E.S.Kuh (1982): Efficient algorithms for channel routing, IEEE Trans. CAD, vol. CAD-1, No1, pp. 25-35

Proc.IMYCS '86 October 13-17,1986
Smolenice Castle,ČSSR

GRAMMARS OF SYNTACTICAL FUNCTIONS AND PROGRAMMING IN LOGIC

U. Lämmel

Wilhelm-Pieck-Universität
Sektion Informatik
Albert-Einstein-Str. 21
GDR 2500 Rostock
German Democratic Republic

0. Introduction

Attribute grammars are a tool for the definition of programming languages and are widely used in compiler construction especially in Translator Writing Systems.
During the last years some results about the relationship between attribute grammars and logical programming have been published (Deransart [2], Forbrig [3]).
In the paper we will use the relationship and suggest an implementation of a translator for a language as a logical program.
We suppose that the language is defined by a Grammar of Syntactical Functions (see Riedewald [4]), a special form of attribute grammar. The way of implementation will be explained by an example:
A context-free grammar defines the syntax of an arithmetic expression:

```
S    : EXPR,'='.              FAC : '(',EXPR,')'.
EXPR : EXPR,ADD,TERM.         FAC : 'INTEGER'.
EXPR : TERM.                  ADD : '+'.
TERM : TERM,MUL,FAC.          ADD : '-'.
TERM : FAC.                   MUL : '*'.
                              MUL : '/'.
```

Terminals are enclosed in quotation marke, ',' denotes the sequence, ':' separates left-hand side from right-hand side

and the point marks the end of a rule. The terminal INTEGER represents all possible integer-numbers.
The interpretation of an expression is the calculation of a value according to the arithmetic rules: input 3*(4+1)= produces the result 15.

1. Grammar of Syntactical Function

A GSF allowes the explicit definition of language-semantics. Attributes can be assigned to nonterminals as well as to terminals and semantic functions can be added to the rules. An attributed variant of our example-grammar is:

```
S           : EXPR(val),'=' & OUTPUT(x,val).
EXPR(val)   : EXPR(val1),ADD(op),TERM(val2)
                              & ADDSUB(val,val1,val2,op).
EXPR(val)   : TERM(val).
TERM(val)   : TERM(val1),MUL(op),FAC(val2)
                              & MULDIV(val,val1,val2,op).
TERM(val)   : FAC(val).
FAC(val)    : 'INTEGER'(val).
FAC(val)    : '(',EXPR(val),')'.
ADD("+")    : '+'.            MUL("*") : '*'.
ADD("-")    : '-'.            MUL("/") : '/'.
```

Attributes are separated by "," and attribute lists are enclosed in brackets. Semantic functions are written after '&'.

The same attribute name contains the same value. Attribute names have only local meaning within a rule. The semantic function ADDSUB (MULDIV) calculates the new value 'val' of an expression (term) by adding or subtracting the values of the right-hand-side-expression (term) 'val1' and the Term (factor) 'val2' according to the operation 'op'. The usual interpretation of an input starts with the lexical analysis and continues with the construction of a syntax tree. The syntax tree will be decorated by attributes according to some attribute evaluation algorithm and a sequence of calls of semantic functions will be produced. Looking at the string '3*(4+1)=' the call of the functions

```
   ADDSUB(val,4,1,+),
   MULDIV(val1,3,val,*),
   OUTPUT(x,val1)
```

will calculate the semantics.

2. Transformation of a GSF into a PROLOG-program

The relationship between attribute grammars and logical programming has been mentioned in several papars. Most important is, that there exists a semantically equivalent set of Horn-clauses for a given attribute grammar and vice versa, as shown by Deransart, Maluszinski [2].
The notations of a GSF and a set of PROLOG-clauses are very similar. So questions arise: What has someone to do if he wants to transform a GSF-language description into a logical program? Is it enough to change only the few characters like ":" into ":-" and to omitt the "&"?

A closer look at the example grammar shows, that there are some more problems. Our grammar contains only attributes defining some language-semantics, because in normal translators, based on attribute grammars, lexical and syntactical analysis is done by predefined algorithms. An appropriate PROLOG-program must fulfil all tasks of a translator - lexical analysis, parsing, semantical analysis and interpretation. Hence, our GSF should explicit define even the handling (read-in and check) of terminals. What kind of GSF-possibilities is better qualified for this purpose then attributes and semantic functions!
The next paragraphs give an impression about the necessary modifications of a GSF, in order to become translatable into a logic program.

2.1 Modification of the GSF

2.1.1 Lexical Analysis

A semantic function NEXTINPUT will be introduced to read in the next terminal from input. This function must be called every time, a terminal is recognized by the parsing algorithm.So we

have to introduce a new nonterminal for each terminal. All occurences of the terminal will be replaced by the new nonterminal and a new rule containing the semantic function NEXTINPUT has to be added to the grammar:
'(' is replaced by OPBRACK and OPBRACK gets two attributes 'cur','new' and a rule
OPBRACK("(",new) : '(' & NEXTINPUT(new). is added.
The terminal, read in by NEXTINPUT, must be visible at every place, the new terminal has to checked, wether it is right or not in the current syntactical environment. That's why we have to organize the transport of the terminal. This is done by attributes. Two new attributes are assigned to each terminal and nonterminal, one for the current and one for the next terminal:

```
FAC(cur,new,val) : OPBRACK(cur,new1),EXPR(new1,new2,val),
                                        CLBRACK(new2,new).
```

Terminals, which occure only once as a single element on the right-hand-side of a rule, needn't to be replaced by nonterminals.
A look at terminals like INTEGER shows, that the check, wether the current terminal is an integer or not, can't be done by replacing the attribute by a constant (as we did it with "(" for instance). In this case we add a new semantic function (ISINTEGER) to the rule checking the property.

2.1.2. Parsing

PROLOG uses a theorem-prover which uses a depth-first-strategy. So the interpretation of clauses is very similar to a LL-syntax analysis. Hence, we can use results from the theory of LL-parsing, especially ll(1)-parsing, to avoid useless backtracking. For instance, the left recursivity leads to infinitely analysis and must be avoided. In our example we have to change the rules for EXPR and TERM:

```
EXPR  : TERM,EXPR1.        TERM  : FAC,TERM1.
EXPR1 : ADD,TERM,EXPR1.    TERM1 : MUL,FAC,TERM1.
EXPR1 :   .                TERM1 :   .
```

2.1.3. Semantic processing

The basic attributation in our example defines the calculation of the semantics of a given expression. The nonterminals keep their attributes representing the value of the subexpression. Only the rearrangement of rules causes further attributation of the nonterminals EXPR1 and TERM1. A new attribute is necessary to keep the value of the preceeding subexpressions (TERM/FAC) in order to calculate the new value of the recognized subexpression:

```
EXPR1(cur,new,precval,newval):
                    ADD(cur,new1,op),TERM(new1,new2,val),
                    EXPR1(new2,new,val,val2)
                &   ADDSUB(newval,precval,val2,op).
```

Summarizing the above mentioned steps the modified example GSF has the following form:

```
S : BEGIN(cur), EXPR(cur,new,val), END(new) & OUTPUT(x,val).
EXPR(cur,new,val) :
                TERM(cur,new1,val1),EXPR1(new1,new,val1,val).
EXPR1(cur,new,precval,newval):
                ADD(cur,new1,op), TERM(new1,new2,val),
                EXPR1(new2,new,val,val1)
            &   ADDSUB(newval,precval,val1,op).
EXPR1(cur,cur, val,val):
TERM(cur,new,val):FAC(cur,new1,val1),TERM1(new1,new,val1,val).
TERM1(cur,new,precval,newval):
                MUL(cur,new1,op), FAC(new1,new2,val)
                TERM1(new2,new,val,val1)
            &   MULDIV(newval,precval,val1,op).
TERM1(cur,cur,val,val):
FAC(cur,new,val): 'INTEGER'(int)
            &   ISINTEGER(val,cur,int),NEXTINPUT(new).
FAC(cur,new,val): OPBRACK(cur,new1)
                    EXPR(new1,new2,val), CLBRACK(new2,new).
OPBRACK("(",new): '(' & NEXTINPUT(new).
CLBRACK(")",new): ')' & NEXTINPUT(new).
```

```
ADD("+",new,"+") : '+' & NEXTINPUT(new).
ADD("-",new,"-") : '-' & NEXTINPUT(new).
MUL("*",new,"*") : '*' & NEXTINPUT(new).
MUL("/",new,"/") : '/' & NEXTINPUT(new).
BEGIN(new) : & NEXTINPUT(new) . END("=") : .
```

2.2 A PROLOG-program

After modification a 1-1-mapping of GSF-rules into PROLOG-clauses is possible. Only ortographical changes, depending on the PROLOG implementation, are left. The changes to realize the core-PROLOG notation are:

- Erase all terminals from the rules.
- If the right-hand-side of a rule is empty, even after omitting the terminals, then take away the "&" otherwise replace it by ",".
- If the right-hand-side contains neither nonterminals nor semantic functions then omitt the ":" otherwise replace it by ":-".
- Let the names of variable attributes start with upper-case letters and write the nonterminals and names of semantic functions with small letters.

Some example should be sufficiently to show the form of the resulting set of clauses:

```
s:- begin(Cur),expr(Cur,New,Val),end(New),output(X,Val).
opbrack("(",New) :- nextinput(New).
term1(Cur,Cur,Val,Val).
```

One Problem left: The realization of a semantic function. The semantic functions must be defined by PROLOG-clauses, too. For this task we can't give any recommendation, because it depends on the underlying semantics. In our example we realized the semantic functions by built-in-predicates: e.g.:

```
output(X,Val) :- put(Val).
muldiv(X,Y,Z,"*") :- X is Y*Z.
muldiv(X,Y,Z,"/") :- X is Y/Z.
```

The interpreter for an arithmetic expression is ready. Adding a clause run: -s,run. allowes us to calculate several arith-

metic expressions.

3.Conclusion

Using Grammars of Syntactical Functions modifications were explained in order to allow a 1-1-mapping from GSF-rules into PROLOG-clauses. That means, PROLOG can be considered as a kind of Translator - Writing-System, because it accepts such an attributed grammar, defining a language.
If a GSF-definition of a language exists the implementation as a logic program is mostly a mechanical step as shown in the paper. This method seems to be useful for the development of prototypes of translators.
To use PROLOG as an implementation language for translators of serious languages it should be possible to bind other procedures, written for instance in PASCAL, into a PROLOG-program. An experiment to implement an interpreter for a special command language will start in the next future.

4.Literature

[1] Clocksin,W.F., Mellish,C.S.
Programming in Prolog
Berlin West, Heidelberg, New York, Tokio, 1984.

[2] Deransart,P., Maluszinski,J.
Relating logic programs and attribute grammars
INRIA-Report,1984.

[3] Forbrig,P.
Relation between attributed grammars and Horn clauses
IMYCS'86, in this proceedings.

[4] Riedewald,G., Maluszinski,J., Dembinski,P.
Formale Beschreibung von Programmiersprachen
Berlin,1983

Proc.IMYCS '86 October 13-17,1986
Smolenice Castle,ČSSR

REDUCEDNESS OF FORMAL LANGUAGES

Gy. Lampérth

Institute of Mathematics
Eötvös Loránd University
Múzeum krt. 6-8.
H-1088 Budapest/Hungary

1. INTRODUCTION

The correspondence between sequential program schemes and formal languages is well known, e.g. by Engelfriet [1] . The situation is more complicated in the case of parallel program schemes. Mazurkiewicz [3],[4] has introduced trace languages and Szijártó [6] the so called independence relation to describe them.

In this paper we are introducing the notion of closure of a languge over arbitrary binary relation on the alphabet of the language, see [2] .

An alphabet V is a finite nonempty set. The elements of V are called letters. The finite strings formed from the elements of V are said to be words. The set of all words over an alphabet V is denoted by V^*. Any subset of V^* is a language. Our notations and definitions are the same as those in [5] or [6] .

Definition 1.1. The language $\langle L \rangle_R$ is called the closure of the language $L \subseteq V^*$ over the relation $R \subseteq V \times V$ if

(i) $L \subseteq \langle L \rangle_R$,

(ii) if $w=vabu \in \langle L \rangle_R$ and $(a,b) \in R, v,u \in V^*$ then $w'=vbau \in \langle L \rangle_R$,

(iii) we can get all the elements of $\langle L \rangle_R$ by the rules (i) and (ii).

If $L=\{w\}$, then instead of $\langle L \rangle_R$ we write $\langle w \rangle_R$ for short. The basic properties of the closure of languages can be found in [6] . As it is usual, $\mathcal{L}_i$ denotes the family of type i languages in the Chomsky hierarchy, where i=0,1,2,3.

Definition 1.2. We say that a language L is closable with respect to type i over the binary relation R if there is a type i language L' such that $\langle L' \rangle_R = \langle L \rangle_R$. Here L, $L' \subseteq V^*$ and $R \subseteq V \times V$. The language L' is called the covering system of L.
The properties of closability see in [2] .

2. REDUCED LANGUAGES OF LANGUAGES

Definition 2.1. A language $L \subseteq V^*$ is called a reduced language over a relation $R \subseteq V \times V$ if there is no $w \in L$ such that $w \in \langle w_1 \rangle_R$, where $w_1 \in L$ and $w \neq w_1$.

Theorem 2.1. Let $L \subseteq V^*$ be a language and $R_1, R_2 \subseteq V \times V$ relations. Then

$$\langle L \rangle_{R_1 \cap R_2} \subseteq \langle L \rangle_{R_1} \cap \langle L \rangle_{R_2} .$$

If $R_1=R_2$ or L is a reduced language over V x V, then the equality holds.

Proof. Outline. The inclusion easily follows from $R_1 \cap R_2 \subseteq R_1$, $R_1 \cap R_2 \subseteq R_2$ and $\langle\langle L \rangle_R \rangle_{R'} \subseteq \langle L \rangle_R$, where $R \subseteq R' \subseteq V \times V$. The equality is also clear if $R_1=R_2$. So it is sufficient to prove equality, if L is reduced over V x V. Let $v \in \langle L \rangle_{R_1} \cap \langle L \rangle_{R_2}$. Then there are $v_1, v_2 \in L$ such that $v \in \langle v_1 \rangle_{R_1}$ and $v \in \langle v_2 \rangle_{R_2}$. $v_1=v_2$ follows from reducedness of L over V x V. Let $v'=v_1=v_2$. Then $v \in \langle v' \rangle_{R_1}$ and $v \in \langle v' \rangle_{R_2}$ Both R_1 and R_2 contains those elements (x,y) for which x and y form inversion in v considering v'. These elements (x,y) are also in $R_1 \cap R_2$, so $v \in \langle v' \rangle_{R_1 \cap R_2} \subseteq \langle L \rangle_{R_1 \cap R_2}$.

We have given in [2], a sufficient condition for reducedness of languages from $\mathcal{L}_3$. The condition concerns the rules of grammar generating the language.

Definition 2.2. A formal language L has the language L' as its reduced covering system over the relation R if L' is a reduced language over R and $\langle L\rangle_R = \langle L'\rangle_R$.

Definition 2.3. The language L' is the reduced language of the language L over the relation R, if L' is the reduced covering system of L over R, and $L' \subsetneq L$.

It is obvious from the definitions that the reduced covering system of language L is the reduced language of $\langle L\rangle_R$. It is also clear that a reduced language of a language is, at the same time, the reduced covering system of the language as well. On the other hand, there may exist such a reduced covering system that is not a reduced language - or subset - of L.

Theorem 2.2. Every formal language $L \subseteq V^*$ has a reduced language over an arbitrary relation $R \subseteq V \times V$. If R is antisymmetrical then the reduced language is unique.

Proof. In the language L there is only a finite number of words of the length k for every natural number k,i.e.: there are no more words than n^k where n is the number of letters in the terminal alphabet. The proof will be constructive. The reduced language will be denoted by L'. The words of L will be ranged into a line according to their lengths. We go on step by step: k=0,1,2,... . Of course, k=0 and k=1 are uninteresting. Let the words of length k (consisting of k letters) be $w_1, w_2, \ldots, w_t (t \leq n^k)$

1. $w_1 \in L'$ (conditionally)

2. a. $w_2 \in \langle w_1\rangle_R$ so $w_2 \notin L'$ (independent of $w_1 \in \langle w_2\rangle_R$ or $w_1 \notin \langle w_2\rangle_R$. In the first case there might be $w_2 \in L'$ and $w_1 \notin L'$ as well.

It can be seen that L' is generally not unique. If R is antisymmetrical and if $w_2 \in \langle w_1\rangle_R$ than $w_1 \notin \langle w_2\rangle_R$ in case $w_1 \neq w_2$.

b. $w_2 \notin \langle w_1\rangle_R$. Then two cases are possible.

α. $w_1 \in \langle w_2\rangle_R$, so $w_2 \in L'$ and $w_1 \notin L'$,

β. $w_1 \notin \langle w_2 \rangle_R$ then $w_1, w_2 \in L'$.

3. If either w_1 or w_2 only belongs to L' then w_3 is treated just as w_2 in 2. If both w_1 and w_2 belong to L' then the provision of the general scheme in s. is followed.

s. $w_{i_1}, w_{i_2}, \ldots, w_{i_p} \in L'$ where $2 \leq p \leq s-1 \leq t-1$

a. There exists such w_{i_j} $(1 \leq j \leq p)$, there may be, perhaps, several that $w_s \in \langle w_{i_j} \rangle_R$. Then, in case of $w_s \notin \langle w_{i_h} \rangle_R$ $(i_h \neq i_j)$ no $w_{i_h} \in \langle w_s \rangle_R$ can subsist as, on the contrary, $\langle w_{i_h} \rangle_R \subseteq \subseteq \langle w_s \rangle_R \subseteq \langle w_{i_j} \rangle_R$ is performed, which contradicts to the construction because, in this way, the language $\{w_{i_1}, w_{i_2}, \ldots, w_{i_p}\}$ would not be a reduced one. Consequently, $w_s \notin L'$. If $w_{i_j} \in \langle w_s \rangle_R$ also subsisted - which is impossible in an antisymmetrical case - then there might be $w_{i_j} \notin L'$ and $w_s \in L'$ as well. It can be seen here, too, that L' is generally not unique.

b. For every w_{i_j} $w_s \notin \langle w_{i_j} \rangle_R$ $(1 \leq j \leq p)$. So $w_s \in L'$ and two cases are possible:

α. No w_{i_j} is contained in the closure of w_s, consequently $w_{i_1}, w_{i_2}, \ldots, w_{i_p}, w_{i_{p+1}} = w_s \in L'$.

β. If one - or, perhaps, several - of w_{i_j} is(are) contained in $\langle w_s \rangle_R$, than that(those) is(are) left behind, the indexes of the rest are changed and w_s gets i_p or, if there have been several w_{i_j}'s then it gets index less than i_p.

At the end of the construction we underline that those and only those words belong to L' which did not receive the negation of this statement during the process after $w \in L'$. It is obvious from the construction that L' is unique in case of an antisymmetrical relation.

And this is the end of proof.

Finally we raise two questions: [1] Has any type i language its type i reduced language, or only its type i reduced covering system? [2] Has any language closable with respect to type i its type i reduced language or only a reduced covering system among type i?

R E F E R E N C E S

[1] Engelfriet, J.: Simple program schemes and formal languages, Springer LNCS vol. 20, 1974.

[2] Lampérth,Gy.: Closability and reducedness of formal languages, Papers on Automata and Languages VI. Publ. Dept. of Math. Karl Marx Univ. of Economics, Budapest, 1984, pp. 77-87.

[3] Mazurkiewicz, A.: Concurrent program schemes and their interpretations, Publ. of Inst. of Math. Univ. of Aarhus, 1977.

[4] Mazurkiewicz, A.: Parallel recursive program schemes, LNCS, Vol. 32, Proc. MFCS '75, 1975.

[5] Salomaa, A.: Formal Languages, Academic Press, New York--London, 1973.

[6] Szíjártó, M.: A classification and closure properties of languages for describing concurrent system behaviours, Fundamenta Informaticae, IV:3 (1981).

Proc. IMYCS '86 October 13-17, 1986
Smolenice Castle, ČSSR

FAST FOURIER TRANSFORM: MATRIX- AND ALGEBRAIC APPROACHES

A.Matevosyan

Computing Centre of Armenian SSR
Academy of Sciences
375044, Sevak Str. 1
Yerevan USSR

1.Introduction

Orthogonal transforms are widely applied in various field such as pattern recognition, information theory, signal processing etc. The application of orthogonal transforms is bound up with the problems of two types:

A) constructing of a fast algorithm of calculation of the transforms, which requires $o(N^2)$ operations of multiplication and addition instead of N^2 ,

B) investigation of optimal properties of the transforms in various optimization problems, for example, in the problem of random signal processing.

In this paper the problem A) is considered. So let $\Phi = \{\varphi_n(t)\}$, $n, t \in \Omega = \{0, 1, \dots, N-1\}$, be an orthonormal matrix of order N , $f = (f_0, f_1, \dots, f_{N-1})^T$ be a complex vector. The discrete orthogonal transform of f is the vector $\hat{f} = (\hat{f}_0, \hat{f}_1, \dots, \hat{f}_{N-1})^T$, where

$$\hat{f} = \Phi \cdot f \text{ , or } \hat{f}_m = \sum_{t \in \Omega} f_t \cdot \varphi_m(t) , \; m \in \Omega . \qquad (1)$$

Direct calculation of (1) requires N^2 arithmetic operations (a.o.). Morgenstern [10] has shown that a linear algorithm, computing (1) and using complex numbers α_i , $|\alpha_i| \leq C$, $C > 1/2$, requires greater than $\log |\det \Phi| \; / \; \log 2C$ a.o.

Algorithm of Fast Fourier Transform requires $O(N \sum_{i=1}^{k} N_i) \sim N \log N$ a.o., where $N = N_1 \cdot N_2 \cdot \ldots \cdot N_k$. Thus, the FFT algorithm is optimal since $\det \Phi = N^N$ and $\log |\det \Phi| = N \log N$.

At the present time there are two approaches to solving the problem A). Historically first, matrix approach, consists in constructing of a matrices, having a special structure, and work out a fast algorithm for such matrices. The matrix approach was proposed by Good [5] who has factored the prime product of matrices as a product of sparse matrices.

The second, algebraic approach was developed by Apple and Wintz [1], Nicholson [11], Cairns [3] and Bojko [2]. In these papers the class of transforms on finite abelian groups was considered. The essence of the algebraic approach consists in decomposing of the group (prime sum of subgroups or union of subgroup and cosets) and reducing of the transform to transforms of lesser orders on subgroup and cosets.

The matrix approach has advantages in practical problems of constructing of orthogonal basises with desired characteristics. The algebraic approach permits:

- to construct the complete theory of FFT on abelian groups,
- to generalize, by means of investigation of generalized displacement operators, the theory of FFT on new algebraic object - hypergroups,
- to link the problem A) with the problem B), in which the displacement operators associated with given orthogonal basis play basic role.

2. Matrix approach

If it is possible to factor the transform matrix Φ as a product of sparse matrices

$$\Phi = \Phi_n \cdot \Phi_{n-1} \cdot \ldots \cdot \Phi_0 , \qquad (2)$$

then the calculation of the transform (1) by algorithm

$$f_0 = f , \; f_{i+1} = \Phi_i \cdot f_i , \; i = 0, 1, \ldots, n, \; \hat{f} = f_{n+1} , \qquad (3)$$

will require $\sum_{i=0}^{n} r_i$ a.o., where r_i is the number of non-zero entries of sparse matrix Φ_i . We demonstrate the mat-

rix approach on the several constructions.

2.1. Sum of prime productes

Definition 1. The matrices X and Y of order κ are called X-matrices if

$$X \cdot Y^T = Y \cdot X^T , \quad X \cdot X^T + Y \cdot Y^T = \kappa I_\kappa .$$

Methods of constructing of X-matrices are proposed by Matevosyan [9] .

Theorem 1. Let X, Y be X-matrices of order κ , H_{m_o} be an orthogonal matrix of order $m_o \times m_o$, $Z_p = I_{p/2} \otimes \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$, then the matrices

$$H_{m_n} = X \otimes H_{m_{n-1}} + Y \otimes (Z_{m_{n-1}} \cdot H_{m_{n-1}}) , \; n \geq 1 , \qquad (4)$$

are the orthogonal $m_n \times m_n$ matrices.

Theorem 2. The matrix H_{m_n} , constructed by recurrent formula (4) will be represented in the form (2) of ordinary product of sparse matrices, where

$$\Phi_o = I_{\kappa^n} \otimes H_{m_o} ,$$

$$\Phi_i = I_{\kappa^{n-i}} \otimes (X \otimes I_{m_{i-1}} + Y \otimes Z_{m_{i-1}}) , \quad 1 \leq i \leq n .$$

Each row of matrix Φ_o contains no more m_o nonzero entries, and each row of matrices Φ_i - no more than 2κ , therefore the calculation of transform with the matrix H_{m_n} by algorithm (3) requires no more than $m_n (m_o + 2n\kappa)$ a.o.

2.2. "Flaky prime product"

Matrix is called "flaky prime product", if may be divided into submatrices-layers, each of which is prime product of matrices [7] . The Haar matrices are the classical example of such matrices. The basises of Haar type HA_{m_n} of order $m_n = \kappa_1 \cdot \kappa_2 \cdot \ldots \cdot \kappa_n$ are constructed by recurrent formula

$$HA_{m_1} = B^{(o)}_{\kappa_1} ,$$

$$HA_{m_{n+1}} = \begin{bmatrix} HA_{m_n} \otimes e_{\kappa_{n+1}} \\ \sqrt{m_n} \bigoplus_{i=0}^{m_n - 1} \widetilde{B}^{(i)}_{\kappa_{n+1}} \end{bmatrix} , \; n \geq 1 , \qquad (5)$$

where $e_k=(1,1,\dots,1)$ is k - vector, $\tilde{B}_m^{(i)}$ is the $(m-1)\times m$ matrix, consisting of last $m-1$ rows of matrix $B_m^{(i)}$ of order $m\times m$, $\oplus$ is symbol of prime sum.

Theorem 3. If $B_{\kappa_n}^{(i)}$, $0\le i\le m_n-1$, $n\ge 1$, are orthogonal $\kappa_n\times\kappa_n$ matrices, first row of which consists of +1, then the matrices HA_{m_n} (5) are orthogonal $m_n\times m_n$ matrices.

The classical Haar matrices and k - matrices of Haar, constructed by Izenberg, Rudko and Sisuev [6], are the particular cases of Haar type matrices (5). The construction (5) permits to construct hybrid Haar-Fourier, Haar-Walsh, Haar-Slant etc. basises with desired characteristics.

Theorem 4. The Haar type matrices HA_{m_n} will be represented in the form of ordinary product of sparse matrices

$$HA_{m_n}=M_1\cdot M_2\cdot\dots\cdot M_n,$$

where

$$M_n=\begin{bmatrix} I_{m_{n-1}}\otimes e_{\kappa_n} \\ \sqrt{m_{n-1}}\ \bigoplus_{j=0}^{m_{n-1}-1}\tilde{B}_{\kappa_n}^{(j)} \end{bmatrix},$$

$$M_{i+1}=\begin{bmatrix} I_{m_i}\otimes e_{\kappa_{i+1}} & O \\ \sqrt{m_i}\ \bigoplus_{j=0}^{m_i-1}\tilde{B}_{\kappa_{i+1}}^{(j)} & O \\ O & I_{m_n-m_{i+1}} \end{bmatrix},\quad 0\le i\le n-2.$$

Each row of matrix M_n contains no more than κ_n nonzero entries, and each from first m_{i+1} rows of matrices M_{i+1} contain no more than κ_{i+1} nonzero entries, therefore the calculation of $\hat{f}=HA_{m_n}\cdot f$ requires no more than $\sum_{i=1}^{n} m_i\kappa_i\sim m_n$ a.o.

3. Algebraic approach

3.1. Fourier transforms on abelian groups (Apple and Wintz [1])

Let G be an any finite abelian group of order N,

$\chi(g)$- character of G , $G^\vee$ - group of characters of G . The function $\hat{f}: G^\vee \to \mathbb{C}$, defined by the formula

$$\hat{f}(\chi) = \sum_{g \in G} f(g) \cdot \overline{\chi(g)} , \tag{6}$$

is called the Fourier transform of a function $f: G \to \mathbb{C}$. Direct calculation of the Fourier transform on abelian group of order N requires N^2 a.o.

Let the group G of order $N = N_1 \cdot N_2$ is represented in the form of prime sum of subgroups G_1 and G_2 of order N_1 and N_2 respectively, then $\chi(g) = \chi_1(g_1) \cdot \chi_2(g_2)$, where χ_i are the characters of G_i , $g = g_1 + g_2$, $g_i \in G_i$. The function $f(g)$ can be represented as a function $f(g_1, g_2)$ of two variables, then the Fourier transform on group

$$\hat{f}(\chi) = \sum_{g_2 \in G_2} \Big(\sum_{g_1 \in G_1} f(g_1, g_2)\, \overline{\chi_1(g_1)} \Big) \cdot \overline{\chi_2(g_2)} . \tag{7}$$

Thus, the Fourier transform on abelian group of order N is reduced to Fourier transforms on subgroups of order N_1 and N_2 , therefore the calculation of $\hat{f}(\chi)$ by the formula (7) requires $N(N_1 + N_2)$ a.o. instead of N^2 .

3.1. Fourier transform on commutative hypergroups

Let A^s be a family of commutative generalized displacement operators (GDO) (Levitan [8]): $A^s: \mathbb{C}^N \to \mathbb{C}^N$, $s \in \Omega$, then the set Ω is called a hypergroup (Dunkl [4]). A nonzero complex function $p: \Omega \to \mathbb{C}$ is called a character if the following formula holds:

$$A^s p(t) = p(s) \cdot p(t) .$$

The set of all characters will be denoted by $\Omega^\vee$. The function $\hat{f}: \Omega^\vee \to \mathbb{C}$, defined by the formula

$$\hat{f}_n = \sum_{t \in \Omega} f(t) \cdot \overline{p_n(t)} , \tag{8}$$

is called the Fourier transform of the function $f: \Omega \to \mathbb{C}$ on hypergroup Ω .

The equation for characters one can write in the form

$$\sum_{r \in \Omega} a(s,t,r)\, p_n(r) = p_n(s) \cdot p_n(t) .$$

The subset $\Omega_0 \subset \Omega$ is called a subhypergroup if for each

$s,t \in \Omega_o$ $a(s,t,r)=0$ when $r \notin \Omega_o$. The subset $\Omega_1 \subset \Omega$ is called a coset if for each $t \in \Omega_o$, $s \in \Omega_1$ $a(s,t,r)=0$ when $r \notin \Omega_1$. Denote $\Omega^1=\{0,1,\dots,N_1-1\}$, $\Omega^2=\{0,1,\dots,N_2-1\}$.

Theorem 5. Let Ω_o be a subhypergroup of the Ω and Ω_i, $i=1,2,\dots,N_2-1$, be cosets such that

$$\Omega=\bigcup_{i\in\Omega^2}\Omega_i\ ,\quad \Omega_i\cap\Omega_j=\emptyset\ ,\quad |\Omega_i|=N_1\ ,\quad i\in\Omega^2\ ,$$

and for each $n \in \Omega$ the characters $p_n(t)$ is not equal to zero function when $t \in \Omega_i$, then there exist the systems of orthonormal functions

$$\{p_n^s(t)\}\ ,\quad n,t\in\Omega^1\ ,\quad s\in\Omega^2\ ,$$

and the mapping

$$m:\ \Omega\times\Omega^2 \xrightarrow{onto} \Omega^1$$

such that for each $t \in \Omega_s$

$$p_n(\cdot)=a_n^s\,p^s_{m(n,s)}(\cdot)\ .$$

A function $f(t)$, $t \in \Omega$, can be represented in the form of a function $f(s,r)$ of two variables $s \in \Omega^2$ and $r \in \Omega_s$. The Fourier transform on hypergroup Ω of the function $f(t)$

$$\hat{f}_n=\sum_{s\in\Omega^2}\overline{a}_n^s\sum_{r\in\Omega_s}f(s,r)\,\overline{p^s_{m(n,s)}(r)}=$$

$$=\sum_{s\in\Omega^2}\overline{a}_n^s\,F(m(n,s))\ ,\quad n\in\Omega\ . \qquad (9)$$

Calculation of $F(m(n,s))$, $s \in \Omega^2$, requires $N_2 \cdot N_1$ a.o. and calculation of

$$\sum_{s\in\Omega^2}\overline{a}_n^s\,F(m(n,s))\ ,\quad n\in\Omega\ ,$$

requires $N \cdot N_2$ a.o. Therefore if the conditions of Theorem 5 hold then the calculation of the Fourier transform on hypergroup of order $N=N_1 \cdot N_2$ by the formula (9) requires $N(N_1+N_2)$ a.o.

References

1. Apple G., Wintz P. Calculation of Fourier transform on finite abelian groups. IEEE Trans. Inform. Theory, 1970, v. 16, 2, 233-234.

2. Bojko L. The generalized Fourier-Haar transform on finite abelain group. Digital image processing and applications, "Nauka", 1981, 12-22.
3. Cairns T. On the fast Fourier transform on finite abelian groups. IEEE Trans. Comput., 1971, v. 20, 5, 569-571.
4. Dunkl C. The measure algebra of a locally compact hypergroup. Trans. Amer. Math. Society, 1973, v.179, 331-348.
5. Good I.J. The interaction algorithm and practical Fourier analysis. J. Royal Stat. Soc., 1958, Ser. B 20, 361-372.
6. Izenberg N., Rudko V., Sisuev E. Funct ions and discrète Haar transforms. Izvestija Akad. nauk SSSR, Ser. Techn. Cybern., 1975, 6.
7. Jaroslavski L. Some questions of the theory of discrete orthogonal transforms of signals. Digital image processing and applications, "Nauka", 1981, 33-71.
8. Levitan B. Theory of generalized displacement operators, "Nauka", 1973.
9. Matevosyan A. Complex Hadamard transform and connected questions. Progress in Cybernetics and System Research, 1980, v. 8B, 323-328.
10. Morgenstern J. Note on a lower bound of the linear complexity of the fast Fourier transform. J. Assoc. Comp. Math., 1973, v. 20, 2, 305-306.
11. Nicholson P. Algebraic theory of finite Fourier transform. J. Comput. System Sci., 1971, v. 5, 524-547.

Proc.IMYCS '86 October 13-17,1986
Smolenice Castle,ČSSR

MODAL THEORIES INDUCED BY S-INVARIANTS OF PETRI NETS

A. MOSLEMIE
Helsinki University of Technology
Digital Systems Laboratory
Otakaari 5 A,02150 Espoo, Finland

ABSTRACT

The so-called **S-invariants** are vectors calculated from Petri nets. These vectors contain information concerning a class of properties of Petri net, known as safety properties. In the present work, we employ modal logic to express and derive safety properties of Petri nets (**C/E-systems** and **P/T-nets**). The motivation for this work was to develop a system able to answer questions about such properties.

1. Introduction

The aim of this work is to employ modal logic, the logic of possibility and necessity, to express and derive a class of properties of Petri nets, known as safety properties. Information concerning these properties is obtained from a set of vectors, the so-called **S-invariants**, calculated from the given Petri net. As the next step, calculated **S-invariants** are transformed into the logical formulae of a suitable modal system. This transformation is given here in algorithmic level. Logical formulae obtained in this way induce a modal theory containing formulae formalizing the safety properties of the net under consideration.

The motivation for doing this was to develop a system able to answer questions about safety properties of Petri

nets, and in that way, corresponding properties of systems modelled by Petri nets. Decisions of the type "yes", "no" or "unknown" made, are based on the information contained in the logical formulae obtained from the calculated **S-invariants.**

Two classes of Petri nets, known as Condition/Event-systems (**C/E-systems**) and Place/Transition-nets (**P/T-nets**), are considered. A propositional and a quantified modal system provide logical bases for the theories induced by **S-invariants** of **C/E-systems** and **P/T-nets**, respectively. Propositional modal systems employed for **C/E-systems** are decidable and there are mechanical theorem provers available for them. A mechanical theorem prover will be a part of our question answering system. However, since quantified modal systems are not decidable, situation is not that easy for **P/T-nets.**

A rough description of the architecture of the system is shown in Figure 1.1.

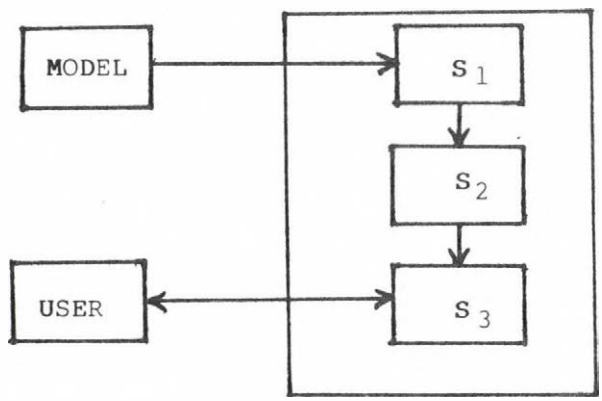


Figure 1.1. The architecture of the question answering system based on **S-invariants.** S_1 computes **S-invariants** of **C/E-systems** and **P/T-nets**, S_2 generates modal theories, and S_3 is composed of a mechanical theorem prover for **C/E-systems** and an implementation of a positive test for **P/T-nets.**

2. MODAL LOGIC, BASIC CONCEPTS

This section contains a concise introduction to modal logics, modal propositional calculus (**MPC**) and modal first order predicate calculus (**MFOPC**). The reason, why modal logic is needed is that, classical logics, propositional calculus (**PC**) and first order predicate calculus (**FOPC**), are not rich enough to express the concepts POSSIBILITY and NECESSITY.

The LANGUAGE of a logical calculus is defined to be the set of FORMULAE constructed from the PRIMITIVE SYMBOLS using a set of FORMATION RULES. Primitives of **MPC** consist of a set of symbols interpreted as atomic formulae, logical connectives, and modal operators **M** (POSSIBILITY) and **L** (NECESSITY). Primitives of **MFOPC** consist of a set of predicate symbols, a set of individual variable symbols, logical connectives, and modal operators **M** and **L** (see [3] for more details).

A Hilbert-type PROOF THEORY of a logical calculus consists of a set of formulae called AXIOMS and a set of TRANSFORMATION RULES. A formula is called DERIVABLE from a set of formulae iff it can be deduced from the axioms and the given set using transformation rules. A formula is a THEOREM (i.e. PROVABLE) iff it is derivable only from the axioms. Any set of formulae closed under deduction is called THEORY.

SEMANTICS of a logical calculus consists of a set-theoretical structure called MODEL, and the appropriate interpretation of the formulae in such a way that they talk about phenomena of the model. A model for a classical logic, **PC** or **FOPC**, consists of a single world (formalizing a state of affairs), where any formula has a single truth-value. Thus, classical logics can consider a world at a time. This kind of formalism is not adequate for our purpose.

The semantics given for modal logic is based on the so-called "MANY-WORLDS" notion. That is, roughly speaking, a modal model consists of a set of worlds **W**, a binary accessability relation **R** over the elements of W, and an interpretation function. A formula may have different truth-values in different worlds in the same model. Thus, modal logic is able to consider many worlds simultaneously (see [3] for more details).

We take as a world, a state of a system (i.e. a marking of the corresponding Petri net). Since systems have different states as the result of their internal state transitions, the applied modelling formalism must be able to consider different worlds (i.e. states) simultaneously. This explains our motivation for employing modal logic as such a formalism.

3. ALGORITHM FOR C/E-SYSTEMS

In this section we give an algorithm for transforming **S-invariants** of **C/E-systems** into a set of formulae of **MPC**.

Step 1 read the set of **S-invariants** I, and the initial marking m_0; put $Z=\emptyset$;

Step 2 choose $i \in I$, compute $k=i'.m_0$ and put $I=I-\{i\}$;

Step 3 construct vector j from i by eliminating zero-components of i;

Step 4 construct following set:
$V=\{(x_1,\ldots,x_{dim(j)}) : x_l \in \{0,1\}$ for $l=1,\ldots,dim(j)\}$; put $W=\emptyset$ and $S=\emptyset$;

Step 5 choose $v \in V$, compute $g=j'.v$ and put $V=V-\{v\}$;

Step 6 if $g=k$ then put $W=W \cup \{v\}$;
if $V \neq \emptyset$ then go to Step 5; otherwise continue from Step 7;

Step 7 choose $w \in W$; put $W=W-\{w\}$; construct logical formula corresponding to w as follows: form a

conjunction P, where the conditions corresponding to the zero-components of w appear negated, and the conditions corresponding to non-zero-components appear un-negated; put S =S U {P};

Step 8 if $W \neq \emptyset$ go to Step 7; otherwise put $Z=Z \cup \{ \bigvee_{s \in S} s \}$ and continue from Step 9;

Step 9 if $I \neq \emptyset$ then go to Step 2; otherwise output the set Z and stop.

Formulae given by this algorithm induce a modal theory **Th** containing formulae formalizing safety properties of the **C/E-system** under consideration. Since **MPC** is decidable, given a formula, there exists a procedure to decide whether it belongs to **Th** or not, by deciding whether the formula obtained from the given one ,using **DEDUCTION THEOREM**, is a theorem of the modal system involved. A complete and formal discussion concerning **Th** and the **Deduction Theorem** is out of scope of this paper.

4. ALGORITHM FOR P/T-NETS

In this section we do for **P/T-nets** exactly what we did for **C/E-systems** in the previous section. That is, the **S-invariants** of the **P/T-net** under consideration are transformed into a set of formulae of **MFOPC**. The algorithm is given as follows:

Step 1 read the set of **S-invariants** I and the initial marking m_0; put $Z=\emptyset$;

Step 2 choose $i \in I$, and put $I=I-\{i\}$;

Step 3 compute const=$i'.m_0$ and regard it as a string belonging to the language generated from alphabets {0,1,...,9} using obvious formation rule;

Step 4 construct logical formula corresponding to

vector i as follows:

(i) construct the following set of atomic formulae: $Z= \{P_k(x_k): k=1,...,dim(i)\}$;

(ii) extract the following subset from Z: $S= \{P_k(x_k) : i_k \neq 0 \}$;

(iii) construct the following sets of strings: $S_1 = \{ (Ex_j) :$ index j appears in S $\}$; $S_2 = \{i_j.x_j+ :$ index j appears in S $\}$;

(iv) form the concatenation of the elements of the sets S_1 and S_2, and denote by C_1 and C_2, respectively;

(v) put (at the beginning of C_2 and replace the last + appearing in it by the string =const)&;

(vi) form the conjunction P of the elements of the set S;

(vii) assign z the concatenation of the strings C_1, C_2 and (P) respecting the order;

Step 5 put $Z=Z \cup \{z\}$;

Step 6 if $I \neq \emptyset$ go to Step 2; otherwise output Z and stop.

Since **MFOPC** is not decidable, the situation is not that easy for **P/T-nets**. However, there are techniques available, not to be discussed here, for handling this problem also.

5. CONCLUSIONS

As a result, a question answering system (based on **S-invariants**) able to decide whether a property expressed by a logical formula is a safety property of the net under consideration or not, can be implemented. It contains three sub-systems. The sub-systems for computing **S-invariants**, and a mechanical theorem prover for the propositional modal system involved, are implemented by Kujansuu **[4]** and

Tuominen [6], respectively, and are available. The case of **P/T-nets** can be handled using available techniques, a positive test for provability, of Fitting [2]. The third sub-system, where modal theories are generated, is obtained by implementing algorithms which perform the transformation of **S-invariants** into the axioms of the **Th**.

There are some related works on the subject, e.g. the work of Clarke [1]. Finally, as an extension to the present work, one may consider a corresponding system for the so-called "high level" Petri nets. An interesting class of such nets, defined by Reisig [5], is known as Relation nets.

REFERENCES

[1] Clarke, E.M., Synthesis of resource invariants for concurrent programs. Trans. Program Lang. & Syst. 2(1980)3, pp. 338-358.

[2] Fitting, M., Proof methods for modal and intuitionistic logics. Dordrecht, D. Reidel publishing company, 1983. pp. 345-353.

[3] Hughes, G.E. and Cresswell, M.J., An introduction to modal logic. London, Methuen and Co. Ltd, 1972. pp. 22-210.

[4] Kujansuu, R., Petri-verkko-analysaattori [An analyser for Petri nets]. Espoo 1982. Helsinki University of Technology, Digital Systems Laboratory, Raportti PVA 1. 125p.

[5] Reisig, W., Petri Nets, An Introduction. Berline, Springer-Verlag, 1985. pp. 111-138.

[6] Tuominen, H., Modaalilogiikka S4:n ratkeavuusalgoritmin toteutus [Implementation of a decision procedure for modal logic S4], master's thesis. Helsinki University of Technology, Department of electrical engineering. Otaniemi 1982. 67 p.

Proc. IMYCS '86 October 13-17, 1986
Smolenice Castle, ČSSR

STRATEGIES OF FILE REDUNDANCY IN AUTOMATED CONTROL SYSTEMS

B.Yu. Natkovich, A.B. Shelkov

Institute of Control Sciences
117342 Moscow
USSR

There is a possibility of the file destruction during the operation of the information systems and that leads to a considerable physical losses because of errors in output results, increase of task solving time, and lastly, impossibility to obtain necessary results.

In order to diminish the file destruction losses is used a file redundancy, which demands some additional resources [1-4].

At present the following three strategies of file redundancy are used [1-2]:

I. The first strategy, to duplicate a file up to some copies. If a main file is destroyed, then the first copy of it is utilized; in the case of its destruction the next copy is utilized and so on.

II. The second strategy uses the peculiarities of organization of current file renovation. These peculiarities consist in preserving its generation history (predecessor files with their updates) instead of current file copies.

If a current file is destroyed we have to go at least one generation back and use the predecessor file with its update file and regenerate the successor file. This backup process would be continued if the first predecessor is destroyed, and it would be necessary to go back another generation and regenerate the predecessor from its own predecessors and so on.

III. The third strategy, a mixed one is based on the first two, whereby copies of the current file and its generation history are preserved. The strategy is to use the copies of the current file first, and then, if all copies of the current file are destroyed, to go to generation history and regenerate the successor file.

The efficiency research of above-mentioned strategies was carried out before in assumption that the main and reserve files have the same time and probability characteristics of renovation and utilization. The existing models and method of redundancy do not allow to take into account the possibility of placing main and reserve files on the magnetic carriers with different characteristics, utilization of various organization methods for main and reserve files and presence of different characteristics of current file generations (volume, renovation time) and so on.

The purpose of this paper is to research strategies of file redundancy, which take into account the differences in time and probability characteristics of renovation of different generation of the current file; besides that we propose it possible to keep not only the copies and the generation histories of main file, but also the copies of generation histories (dumps), that is broadly used in practice. This secures the analysis of redundancy methods which are more adequate to real systems.

Strategy I. This strategy of redundancy could be used for constant files and variable files. Conventionally, we agree to refer the program modules to the constant files.

The main file F_{oo} is reserved by κ copies $F_{o\tau}$, $\tau = \overline{1,\kappa}$. The probability that the file $F_{o\tau}$ will remain undestroyed during the time interval θ_τ while it is being used is ρ_τ. The probability that it will be destroyed during the renovation process is $q_{\rho\tau} = 1 - \rho_\tau$. The probability that a file will be destroyed during the storage is assumed to be zero.

The probability process of system operation during

the solution of renovation can be expressed with

$$\sum_{j=0}^{K+1}\prod_{S=0}^{j} q_{s-1}\,p_j = 1\,, \qquad \text{where } p_{K+1} = q_{-1} = 1\,.$$

The probability that the task will be completed with K copies is:

$$P_I = 1 - \prod_{m=0}^{K} q_m$$

We interpret index I as associated with the first strategy of redundancy.

Let τ_j be the creation time of file copy with index j. Then the operational planning time for the first strategy, on the average, would be:

$$E[T_I] = \sum_{j=0}^{K}\left(\sum_{i=0}^{j}\theta_i\prod_{s=0}^{j} q_{s-1}\,p_j\right) + \sum_{n=0}^{K}\theta_n\prod_{m=0}^{K} q_m + P_I\sum_{j=1}^{K}\tau_j \qquad (\text{with } q_{-1}=1)$$

Strategy II. "Random Walk" theory [5] is found to be applicable to the research of the second strategy.

Initially, there are y predecessors and update files of a file $\varphi_0 : \varphi_1, \varphi_2, \ldots, \varphi_y$ and is needed to creat a new file φ_{-1}.

Let the renovation time of predecessor φ_z be θ_z, $z = \overline{0,y}$

The file φ_z could be renovated (a new file φ_{z-1} could be created) with the probability p_z and it could be destroyed with the probability $q_z = 1 - p_z$ during the time interval θ_z. The process continues until either file φ_0 and all of the predecessors of it are destroyed or the new file is created.

The task is to determine the probability of successful renovation of file φ_0 and average operational time of computer for the second strategy of redundancy.

We'll interpret the motion of the file as the motion of a "particle" on the horizontal i-axis. At time 0 particle is at its initial position $i=0$ and then it moves a unit step in the right or left direction depending on whether the corresponding trial resulted in success or failure, that is "Random Walk". The work of this system terminates

when the particle reaches one of absorbing barriers ($i=-1$ or $i=y+1$). Let v_i be the probability that the file φ_i and all the predecessor files will be destroyed and ω_i the probability that the file will be renovated. In "Random Walk" terminology v_i and ω_i are the probabilities that a particle which starting at i will be absorbed at $i=y+1$ and $i=-1$, respectively.

Therefore, the probability process of system operation when the second strategy is used can be expressed by the following difference equation

$$\omega_i = q_i \omega_{i+1} + p_i \omega_{i-1}, \quad i=\overline{0,y} \qquad (I)$$

with the boundary conditions $\omega_{-1}=1, \omega_{y+1}=0$ (2). It is known that the solution is given as

$$\omega_i = \frac{\sum_{j=i}^{y} \prod_{k=0}^{j} \frac{p_k}{q_k}}{1+\sum_{j=0}^{y} \prod_{m=0}^{j} \frac{p_k}{q_k}}$$

Therefore, the probability of file φ_0 successful renovation is determined as

$$p_{ij} = \omega_0 = \frac{\sum_{j=0}^{y} \prod_{k=0}^{j} \frac{p_k}{q_k}}{1+\sum_{j=0}^{y} \prod_{k=0}^{j} \frac{p_k}{q_k}}$$

and the probability of the file φ_0 destruction and destruction of all its predecessors is

$$v_0 = \frac{1}{1+\sum_{j=0}^{y} \prod_{k=0}^{j} \frac{p_k}{q_k}}$$

Let us consider the time characteristics of the process when the second strategy is used.

Let t_j be the average duration of computer operation independently of whether the task is successfully solved or not. In "Random Walk" terminology t_j is the time until particle starting at point j will be absorbed at $j=-1$ or $j=y+1$. Therefore the time process of system operation when the strategy II is used could be expressed by the following difference equation:

$$t_j = \theta_j + q_j t_{j+1} + p_j t_{j-1}, \quad j=\overline{0,y} \qquad (3)$$

with the boundary conditions $t_{-1}=0,\ t_{y+1}=0$ (4). By solving this task (3) - (4) we obtain the operational planning time, on the average, for the second strategy of redundancy:

$$E[T_{II}]=t_0=\frac{1}{1+\sum_{i=0}^{y}\prod_{\ell=0}^{i}\frac{p_\ell}{q_\ell}}\sum_{k=0}^{y}\frac{\theta_k\sum_{i=k}^{y}\prod_{\ell=0}^{i}\frac{p_\ell}{q_\ell}}{q_k\prod_{\ell=0}^{k}\frac{p_\ell}{q_\ell}}$$

Strategy III. This is a mix of the previous two strategies, whereby both copies of the current file and its generation history are preserved moreover, the latter in its turn is also reserved by copies (dumps).

Initially, there are $y+1$ generations of the current file in the system: $F_0, F_{-1}, \ldots, F_{-y}$ (F_0 -main file, the rest is its generation history). Let the current file generation with index j has x_j reserve copies ($j=\overline{0,y}$). The probability of successful renovation for the j-th generation and its copies is p_j , the probability of destruction during the renovation is $q_j=1-p_j$; the time of renovation - θ_j, time of duplication - τ_j .

The strategy is to use first the copies of the current file and if all copies of the current file are destroyed, to go to the first, the second, and so on, predecessors of the current file.

With the results of the second strategy of redundancy one could directly state the probability of successful renovation

$$P_{III}=\frac{\sum_{j=0}^{y}\prod_{k=0}^{j}\frac{\tilde p_k}{\tilde q_k}}{1+\sum_{j=0}^{y}\prod_{k=0}^{j}\frac{\tilde p_k}{\tilde q_k}} \qquad (5)$$

and the operational planning time, on the average, for this strategy is:

$$E[T_{III}]=\frac{1}{1+\sum_{i=0}^{y}\prod_{\ell=0}^{i}\frac{\tilde p_\ell}{\tilde q_\ell}}\sum_{k=0}^{y}\frac{\tilde\theta_k\sum_{i=k}^{y}\prod_{\ell=0}^{i}\frac{\tilde p_\ell}{\tilde q_\ell}}{\tilde q_k\prod_{\ell=0}^{k}\frac{\tilde p_\ell}{\tilde q_\ell}}+P_{III}\sum_{i=0}^{y-1}\chi_{i,i+1}\tau_i \qquad (6)$$

where

$$\chi_{i,i+1}=\begin{cases}x_{i+1}-x_i\ , & x_i<x_{i+1}\\ 0\ , & x_i\geq x_{i+1}\end{cases}$$

and $\tilde{p}_k, \tilde{q}_k, \tilde{\Theta}_k$ are determined from expressions

$$\tilde{p}_k = 1 - q_k^{x_k+1}, \quad \tilde{q}_k = q_k^{x_k+1} \tag{7}$$

$$\tilde{\Theta}_k = (1-q_k^{x_k+1})(\Theta_k p_k^{-1} + x_{k-1}\tau_{k-1}) + \tau_k q_k p_k^{-1}\left[1 - q_k^{x_k}(1+x_k p_k)\right], \tag{8}$$

$k=\overline{0,y}$, $x_{-1}=x_0$, τ_{-1} - creation time of renovated file co-ty.

The last addend in (6) defines the time spendings on the recreation of initial reserve structure after the termination of renovation task solving.

The average exploitation expenditures on the creation and conducting of reserve form from computer exploitation expenditures, expenditures on the physical carriers of information and losses in system in the case of the destruction of file and its reserve. These expenditures are determined as:

$$F_i = \left[z_M E[T_i] + z_B(1-p_i)\right]\nu T + (\mu T+1)\,\overline{z}_i, \quad i=\overline{1,3}$$

where

$$\overline{z}_i = \begin{cases} \sum_{\ell=0}^{x} z_{k\ell}, & i=1,2 \\ \sum_{\ell=0}^{y} (x_\ell+1)\, z_{k\ell}, & i=3 \end{cases}$$

where z_M -cost of computer time unit, z_B - losses from destruction of file and its reserve, $z_{k\ell}$ - cost of physical carrier of ℓ-th generation of current file, ν - intensity of inquiry flow on file renovation, μ - intensity of reserve carriers substitutions (generally speaking, μ is an increasing function of ν), T - time interval on which the behavior of system is researched.

On the basis of considered models we worked out a complex of programs for creation and conduction of reserve, realized in language FORTRAN-IV for local display terminal EC-7920 [6]. Among the tasks solved by system are such as calculation and analysis of time, probability and cost characteristics of renovation, construction of characteristics dependence diagrams of different redundancy strategies.

The utilization of this program complex allowes to realize choice of optimal file redundancy methods in the dialogue regime.

Practical recommendations obtained in the result of utilization of this program complex were applied to the project of several information systems and that has allowed to rise a reliability of systems, on the average, for 10-15%.

REFERENCES

1. Кульба В.В., Мамиконов А.Г., Шелков А.Б. Резервирование программных модулей и информационных массивов в АСУ. - "Автоматика и телемеханика", 1980, № 8, с. 133-141.
2. Turksen I.B., Kul'ba V.V. Strategies of file redundancy in information systems. - Microelectronics and Reliability, 1980, v.20, No. 1-2, p. 131-144.
3. Мамиконов А.Г., Кульба В.В., Шелков А.Б. Резервирование программного и информационного обеспечения в вычислительных системах. - Национальная конференция с международным участием "Надежность электронно-вычислительных машин и систем". Тезисы докладов, НРБ, 1984 г.
4. Мамиконов А.Г., Кульба В.В., Шелков А.Б. Восстановление работоспособности вычислительных систем при разрушении программ и данных. - Международная конференция "Системы, допускающие неисправности и диагностика". Тезисы докладов, ПНР, 1985 г.
5. Feller V. An Introfuction to probability theory and its applications, Wiley, 1970, vol. 1.
6. Кульба В.В., Мамиконов А.Г., Наткович Б.Ю., Шелков А.Б. Диалоговая система выработки рекомендаций по резервированию модулей и массивов в АСУ. - "Приборы и система управления", 1985, № 6, с. 7-8.

Proc. IMYCS '86 October 13-17, 1986
Smolenice Castle, ČSSR

THE EFFICIENCY OF THE DEPTH FIRST ALGORITHM FOR RANDOM BOOLEAN MATRICES

Daniel Olejár

Department of Theoretical Cybernetics, Faculty of Mathematics and Physics, Comenius University, 842 15 Bratislava
Czechoslovakia

Abstract. The theoretical efficiency of the depth first algorithm for random Boolean matrices was estimated.

Introduction.

Boolean matrices are often used as data structures in data processing. There were some methods of data processing developed specially for Boolean matrices. The depth-first-algorithm (DFA) is one of those methods. It was introduced in [3] some questions concerning its implementation and modification can be found in [2].

The DFA is a data compression method. It was shown [3] that the DFA is a very useful method in processing of visual images. However, according to [4] almost all Boolean matrices are practically imcompressible by means of the DFA. Therefore it is important to study various classes of Boolean matrices to specify the area of its effective application.

The theoretical efficiency of the DFA will be studied in this paper for random square Boolean matrices and the area of practical applicability of the DFA will be established.

Preliminaries.

We shall consider square Boolean matrices A_N of the type N x N, $N = 2^n$ (n an integer) in this paper. The (i,j) entry of A_N is denoted by $a_{i,j}$. The class of all square Boolean matrices A_N will be denoted by $\mathcal{M}_N$.
The symbol $\sigma(n,m)$ denotes the n-bit binary number m. The submatrix $A^{k,\sigma}$ of A_N ($\sigma = \sigma(2n-2k,m)$, $m=0,1,\ldots,2^{2n-2k}-1$)

$$A^{k,\sigma} = \{(i,j);\ 0 \leqslant i = r, r+1, \ldots, u \leqslant N-1,\ 0 \leqslant j = s, s+1, \ldots, v \leqslant N-1\}$$

will be called normal submatrix of rank k, if the following two conditions are satisfied

1. $u - r = v - s = 2^k$,
2. $(r = 0) \bmod 2^k$, $(s = 0) \bmod 2^k$.

There are two important cases of above definition. The first one is when k = n, i.e. A_N is a normal submatrix of rank n. The second special case is when k = 0. That means that every element of A_N is a normal submatrix of rank 0. In every case ($0 \leqslant k \leqslant n$) A_N consists of 2^{2n-2k} disjoint normal submatrices of rank k.
We shall say that $A^{k,\sigma}$ covers the element (i,j) if $(i,j) \in A^{k,\sigma}$. Every normal submatrix $A^{k,\sigma}$ ($k > 0$) can be decomposed into four normal submatrices of rank k-1 in the following way (Figure 1.). The DFA is based on a decomposition

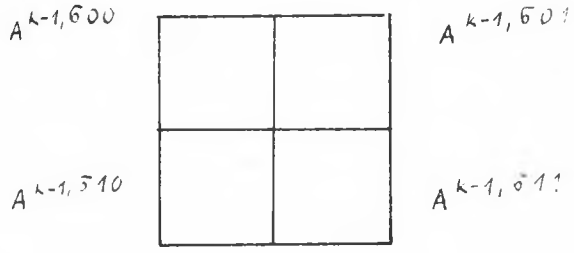


Figure 1.

of a normal submatrix into normal submatrices of lower ranks, all elements of which have the same value. Such normal submatrices will be called constant normal submatrices (c.n.s.) and will be denoted by $A^{k,\sigma}(c)$ (or $A^{k,\sigma}(1)$, $A^{k,\sigma}(0)$ if we need to distinguish between 1-normal submatrices and 0-normal submatrices). We shall give a simplified description of the DFA now. For full description see [3].

The depth first algorithm

There exists the unique DF-expression df $A^{k,\sigma}$ for every normal submatrix $A^{k,\sigma}$ $(0 \leqslant k \leqslant n,\ \sigma = \sigma(2n-2k,m))$ which can be constructed in the following way (Figure 2.)

1. if $A^{k,\sigma} = A^{k,\sigma}(0)$ then df $A^{k,\sigma} = 0$,
   if $A^{k,\sigma} = A^{k,\sigma}(1)$ then df $A^{k,\sigma} = 1$; else
2. df $A^{k,\sigma}$ = (df $A^{k-1,\sigma 00}$)(df $A^{k-1,\sigma 01}$)(df $A^{k-1,\sigma 10}$) (df $A^{k-1,\sigma 11}$).

If k = 1, the parentheses in df $A^{1,\sigma}$ are omitted. According to [3] the right parentheses of DF-expression can be omitted too.

A_8

0	0	0	0	1	1	1	1
0	0	0	0	1	1	1	1
0	0	0	0	1	1	1	1
0	0	0	0	1	1	1	1
0	0	1	1	0	0	1	0
0	0	1	1	0	0	0	1
1	1	0	0	1	1	1	1
1	1	0	0	1	1	0	1

df A_8 =

01(0110(0(1001(1(1101

Figure 2.

We shall deal with random Boolean matrices in this paper. Boolean matrix A_N is called random Boolean matrix if

$$a_{i,j} = \begin{cases} 1 & \text{with the probability } p \\ 0 & \text{with the probability } 1-p, \end{cases}$$

$i,j \in \{0, \ldots, N-1\}$ and $a_{i,j}$ does not depend on $a_{i',j'}$ for $(i',j') \neq (i,j)$.

The class $\mathcal{M}_N$ is understood as a probability space with a probability measure P defined as follows :
For each $A_N \in \mathcal{M}_N$ we set

$$P(\{A_N\}) = p^t(1-p)^{N^2-t},$$

where t is the number of ones in A_N; for an arbitrary $\mathcal{A} \subseteq \mathcal{M}_N$

$$P(\mathcal{A}) = \sum_{A_N \in \mathcal{A}} P(\{A_N\}).$$

Let $\mathcal{V}$ be a property of Boolean matrices. We say that a random Boolean matrix has the property $\mathcal{V}$ if

$$\lim_{N\to\infty} P(\{A_N \text{ has the property } \mathcal{V}\}) = 1. \qquad (1)$$

(If p = 1/2, we shall say that almost all Boolean matrices have the property $\mathcal{V}$ if (1) holds).

The theoretical efficiency of the DFA for random Boolean matrices.

The complexity of the DF-expression can be defined in various ways. Boolean matrices are usually processing by means of computer and so the natural measure of complexity is the number of bits of memory, which are needed to save the (binary encoded) DF-expression. This, real complexity of df A_N was studied in [3]. It was shown, that the real complexity of df A_N depends mainly on the number of zeroes and ones in df A_N. The number of zeroes and ones in df A_N will be referred to as the theoretical complexity of df A_N and denoted by $C_t(A_N)$. We shall study the theoretical complexity of DF-expressions of random Boolean matrices. The real complexity of df A_N will be briefly discussed in Conclusions.

We shall estimate the parameter $C_t(A_N)$ for random Boolean matrices (p an constant, $p \in (0,1)$) now. We need two following lemmas.

Lemma 1. (Markov's inequality) Let ξ be a positive random variable, M_ξ the expectation of ξ, $t > 0$ then

$$P(\xi \geq t) \leq M_\xi / t. \qquad (2)$$

Lemma 2. (Chebyshev's inequality) Let ξ be a random variable, M_ξ the expectation and D_ξ the variance of the random variable ξ, let $t > 0$, then

$$P(|M_\xi - \xi| \geq t) \leq D_\xi / t^2. \qquad (3)$$

For proofs see Feller [1].

We also need the following formula to express the variance

$$D_\xi = M\xi^2 - (M_\xi)^2. \qquad (4)$$

We shall deal with c.n.s. of rank k (k=k(n)). We compute the expectation and the variance of the number of c.n.s. in random Boolean matrices. Let the random variable $\xi_{N,k,p}$ ($\xi_{N,k,p} : (\mathcal{M}_N, P) \to \{0, \ldots, 2^{2n-2k}\}$) attains the value m for Boolean matrix A_N, where m is the number of c.n.s. of rank k in A_N ($n \geq k \geq 0$).

Lemma 3.

$$M\xi_{N,k,p} = 2^{2(n-k)}(p^{2^{2k}} + (1-p)^{2^{2k}}), \qquad (5)$$

$$D\xi_{N,k,p} = 2^{2(n-k)}(p^{2^{2k}} + (1-p)^{2^{2k}})(1-(p^{2^{2k}} + (1-p)^{2^{2k}})) \qquad (6)$$

Proof. The probability that the concrete normal submatrix of rank k is constant is $(p^K + (1-p)^K)$, where $K=2^{2k}$. There are $2^{2(n-k)}$ independent normal submatrices of rank k in A_N and so

$$M\xi_{N,k,p} = 2^{2(n-k)}(p^K + (1-p)^K).$$

We have computed the value $M\xi_{N,k,p}$ yet and according to (4) we have to do it for the value $M\xi^2_{N,k,p}$. The symbol $M\xi^2_{N,k,p}$ denotes the expectation of the number of ordered pairs of c.n.s. of rank k in random Boolean matrices. We shall compute its value.

There are two kinds of ordered pairs of c.n.s. of rank k :

1. with identical items

   The number of such pairs is $2^{2(n-k)}$ and their contribu-

bution to $M\xi^2_{N,k,p}$ is

$$2^{2(n-k)}(2^{2(n-k)}-1)(p^K+(1-p)^K)^2. \qquad (8)$$

Taking into account (7) and (8) we recieve

$$M\xi^2_{N,k,p} = 2^{4(n-k)}(p^K+(1-p)^K)^2 + 2^{(n-k)}(p^K+(1-p)^K). \cdot (1-p^K-(1-p)^K). \qquad (9)$$

By substituting (5) and (9) into 4) we recieve (6).

q.e.d.

We will use the results of Lemma 3. now, to construct the lower and upper bounds of the number of c.n.s. of rank k in random Boolean matrices.

Theorem 1. Let $\xi_{N,k,p}(A_N)$ denotes the number of c.n.s. of rank k in random Boolean matrix A_N, let $\varphi(n)\to\infty$ as $n\to\infty$. Then with the probability tending to 1 as $n\to\infty$ the following inequalities hold :

$$2^{2(n-k)}(p^{2^{2k}}+(1-p)^{2^{2k}}) - \varphi(n).2^{n-k}\sqrt{p^{2^{2k}}+(1-p)^{2^{2k}}} < \xi_{N,k,p}(A_N) < 2^{2n-2k}(p^{2^{2k}}+(1-p)^{2^{2k}}) + \varphi(n).2^{n-k}\sqrt{p^{2^{2k}}+(1-p)^{2^{2k}}}. \qquad (10)$$

Proof. We substitute (5), (6) and

$$t = \varphi(n).2^{n-k}\sqrt{p^{2^{2k}}+(1-p)^{2^{2k}}}$$

into Chebyshev's inequality (3). The value of expression $D\xi_{N,k,p}/t^2$ tends to 0 as $n\to\infty$ and thus the inequality

$$\left|\xi_{N,k,p}(A_N)-2^{2n-2k}(p^{2^{2k}}+(1-p)^{2^{2k}})\right| < \varphi(n).2^{n-k}\sqrt{p^{2^{2k}}+(1-p)^{2^{2k}}}$$

holds with the probability tending to 1 as $n\to\infty$.

q.e.d.

Corollary. Random Boolean matrices contain with the probability tending to 1 as $n\to\infty$

$$2^{2n-2k}(p^{2^{2k}}+(1-p)^{2^{2k}})$$

constant normal submatrices of rank k, k is an constant.

The DFA is a data compression method. The efficiency (theoretical) of the DFA for Boolean matrix A_N is the reciprocal value of the compression coefficient $\varkappa(A_N)$, where $\varkappa(A_N) = C_t(A_N)/N^2$. We shall estimate the theoretical efficiency of the DFA for random Boolean matrices.

Theorem 2. Let $\varepsilon(A_N)$ be the theoretical efficiency of the DFA for random Boolean matrix A_N, then with the probability tending to 1 as $n \to \infty$ holds

$$1/(1 - 3.(p^4+(1-p)^4)/4) < \varepsilon(A_N) < 1/(1 - (p^4+(1-p)^4)). \quad (12)$$

Proof. Random Boolean matrix contains with the probability tending to 1 as $n\to\infty$ $N^2(p^4+ (1-p)^4)(1+o(1))$ c.n.s. of rank 1. (We have substituted n for $\varphi(n)$ in (10)). The c.n.s. of rank 1 cover at least $N^2(p^4+(1-p)^4)(1+o(1))$ elements of random Boolean matrix. The rest of elements is covered by c.n.s. of rank 0 and so, consequently

$$C_t(A_N) < N^2.2^{-2}.(p^4+(1-p)^4)(1+o(1)) + N^2(1-(p^4+(1-p)^4)(1+o(1))$$
$$= N^2.((1-3(p^4+(1-p)^4)/4)(1+o(1)) \quad (13)$$

and the theoretical efficiency of the DFA is not less than $(1 - 3(p^4+ (1-p)^4)/4)^{-1}$.

We shall find the upper bound of $\varepsilon(A_N)$. We have proved that $N^2.(1-(p^4+(1-p)^4)(1+o(1))$ elements of random Boolean matrix is covered by c.n.s. of rank 0. The rest of elements of random Boolean matrix can be covered by c.n.s. of higher ranks. Let us suppose that the number of these c.n.s. is $o(N^2)$. In this case

$$C_t(A_N) > N^2.(1-(p^4+ (1-p)^4)(1+o(1)). \quad (14)$$

The inequalities (13) and (14) hold with the probability tending to 1 as $n\to\infty$ for random Boolean matrix A_N. Our proof is completed.

q.e.d.

Remark. More precise estimation of the theoretical efficiency fo the DFA can be obtained. Taking into account c.n.s. of

rank 2 we have constructed following estimations of $\varkappa(A_N)$

$$1-3(p^4+(1-p)^4)/4 \quad -(p^{16}+(1-p)^{16})/4 < \varkappa(A_N) < 1-3(p^4+(1-p)^4)/4 - 3(p^{16}+(1-p)^{16})/16. \tag{15}$$

Corollary. Almost all Boolean matrices (the case p = 1/2) are practically imcompressible by the DFA.

Conclusions.

The real efficiency of the DFA was studied in [3]. It was proved that if the theoretical complexity of the DF-expression is less than $3.N^2/(4.\log_2 3)$, the real complexity of the DF-expression is less than N^2.

Our results show that if $p \in (0, 0.10003264) \cup (0.89996736, 1)$ random Boolean matrix of $(\mathcal{M}_N, P)$ can be with the probability tending to 1 as $N \to \infty$ effectively processed by means of the DFA.

References.

[1] Feller W.: An Introduction to Probability Theory and its Applications, Vol. 1., Mir, Moscow 1984, pp. 242 - - 248. (in Russian)

[2] Chmúrny J.- Levický D.: Algorithmus for Binary Image Encoding, CAI, Vol. 4., No. I, 1985, pp. 59 - 65.

[3] Kawaguchi E.- Endo T.: On a Method of Binary Picture Representation and its Application to Data Compression. IEEE Transaction on Pattern Analysis and Machine Intelligence, PAMI-2, 1, January 1981, pp. 28 - 65.

[4] Olejár D.: The Efficiency of the Depth First Algorithms CAI, Vol. 5., No. IV., 1986.

[5] Škoviera M.: On the Minimization of Random Boolean Functions, Part II. CAI, Vol. 5. No. IV., 1986.

Proc. IMYCS '86 October 13-17, 1986
Smolenice Castle, ČSSR

METHOD OF ALTERNATIVE FOR KNOWLEDGE REPRESENTATION

S.U.Solowiev, G.M.Solowieva

Institute of Mathematics
Academy of Sciences of MSSR
277028 Kishinev
USSR

The fact that the production rule is the most suitable form for the knowledge representation of expert is widely spread. At the same time the substantial part of expert's knowledge may be represented in the form of alternative or uncompatible statements.

The finite sets (N, Σ, P), where N is the set of attributes, Σ is the set of values, P is the totality of subsets of the form

$$< S_1, S_2, \ldots, S_n >, \quad S_i \in N \times \Sigma, \quad (i = 1, \ldots, n)$$

are define as a system of alternatives, or A-system. Below the pair (A,a) from $N \times \Sigma$ will be called the statement relative to the attribute A and denoted by $A : a$. The elements of set P will be called alternatives.

Let us introduce some definitions for A-system $R = (N, \Sigma, P)$.

$$st(R) = \bigcup_{\alpha \in P} \alpha ;$$

$$neg(R, S) = \{ S' \in \alpha \mid S \in \alpha, \alpha \in P \} \setminus \{ S \}, S \in st(R);$$

$$def(R, A) = \{ X : y \in st(R) \mid X = A \}, A \in N;$$

$$sp(R) = \{ def(R, A) \mid A \in N \}.$$

The A-system is simple or SA-system, if $N \cap \Sigma = \emptyset$ and $sp(R) \subseteq P$.

Let f be characteristic function over the set $st(R)$

$$f : st(R) \to \{0, 1\}.$$

We shall say, that f satisfies to the SA-system R, if for every alternative $< S_1, \ldots, S_n >$ the equality:

$$f(S_1) + \ldots + f(S_n) = 1$$

holds. The totality of characteristic functions over the set $st(R)$, satisfying to the SA-system R, is called class of recognition and denoted by $K(R)$.

The set of statements, on which characteristic function get the value 1, can be considered as description of some object. It is proved, that for the arbitrary set E of object descriptions, the system of alternatives R_E such that the class of recognition coicides with E, can be build. In the general case in order to build the system R_E, it is necessary to enlarge the set of initial basic attributes by new attributes, which play the role of auxiliary variables and have no semantic meaning.

We are interested in studying the possibility and properties of noncontradictory extending of characteristic function, given on some subset of $st(R)$. More exactly, we are interested in the studying the properties of the set

$$K'(R) = \left\{ f \in K(R) \mid f(S) = 1, S \in L_1 \right\},$$

where $L_0, L_1 \subseteq st(R)$, $L_0 \cap L_1 = \emptyset$.

Such situation appears in the dialoge process, when a part of information about the object from the class of recognition is already received. The properties of the $K'(R)$ of the form :

$$\forall g \in K'(R) \qquad g(B : b) = 1 \qquad (1)$$

$$\forall g \in K'(R) \qquad g(C : c) = 0 \qquad (2)$$

relative to new statements $B : b$ and $C : c$, which do not belong to $L_0 \cup L_1$ are of special value. These properties can be easy interpreted : if the object belongs to recognition class then on the base of obtained symptoms may conclude,

that object has the characteristic B : b , but not characteristic C : c.

In the general case in the order to determine properties (1) and (2) it is necessary to receive the family of solutions of integer equations, which define the class of recognition. It is known, that this problem is related to the NP-complete class, therefore we'll define one simple transformation of the system of alternatives, which permits to determine some properties of the type (1), (2).

τ-transformation of the SA-system $R = (N, \Sigma, P)$ relative the set of statements M , is called SA-system $\tau(R,M)$ of the form (N, Σ, P'), where

$$P' = \{ \alpha \setminus M \mid \alpha \in P \} \cup sp(R).$$

The following equality

$$K'(R) = K(R'),$$

where $R' = \tau(R, \bigcup_{S \in L_1} neg(R,S) \cup I$

is true.

As the result of τ-transformation of the SA-system it can appear three particular cases of the alternatives combinations.

Rule 1. If SA-system R' contains empty alternative $< >$, then $K(R') = \emptyset$. (The noncontradictory extending of characteristic function is not possible).

Rule 2. If SA-system R' contains alternative of the form $B : b$, then class of recognition $K(R')$ has the property (1).

Rule 3. If SA-system R' contains alternative

$$\alpha = \langle C : c_1, \ldots, C : c_m \rangle , \quad (m > 0)$$

then for the every statement $C : c$ from $def(R,C) \setminus \alpha$ class of recognition has the property (2).

If the rule 2 and 3 can be applied to the SA-system R' then τ-transformations may be continued. The process of transformations ends after application of the first rule or in the case, when the rule 2 and 3 do not bring new informa-

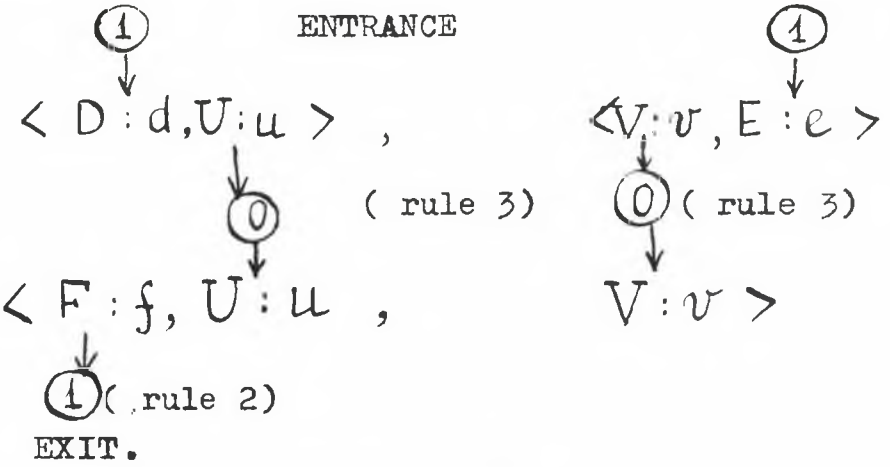
tion about the object of recognition.

Let us the object of consultation possess the properties D : d and E : e, and SA-system contains the alternatives

$$\langle D : d , U : u \rangle , \langle V : v , E : e \rangle ,$$

$$\langle F : f , U : u , V : v \rangle ,$$

then $\tilde{\iota}$-transformation allows to infer new property F : f of the object of consultation



The possibility of adding in the system of alternatives auxiliary attributes permits to enlarge their expressible possibilities.

The cortege of statements :

$$[A_1 : a_1, \ldots, A_n : a_n] \qquad (3)$$

is considered as abbreviated record of the alternative pair

$$\langle A_1 : a_1, \ldots, A_n : a_n, X : x_1 \rangle ,$$

$$\langle X : x_1, X : x_2 \rangle ,$$

where X - new (auxiliary) attribute, which corresponds to cortege (3), x_1, x_2 - some values.

There are some methods of representation of expert knowledges in the system of alternatives of the form

IF $A_1 : a_1$ and ... and $A_n : a_n$ THEN B : b.

For example

R_Y : $[A_1 : a_1, Y : 1], \ldots, [A_n : a_n, Y : n],$

$\langle B : b, Y : 1, \ldots, Y : n \rangle$, $\langle Y : 0, Y : 1, \ldots, Y : n \rangle$,

and

$R_Z : [A_1 : a_1, Z : 1, Z : 1'], \ldots, [A_n : a_n, Z : n, Z : n']$,

$\langle B : b, Z : 1, Z : 1', \ldots, Z : n, Z : n' \rangle$,

$\langle Z : 0, Z : 1, Z : 1', \ldots, Z : n, Z : n' \rangle$.

The system R_Y and R_Z have the same classes of recognition however τ-transformation of the system R_Z does not permit to obtain the property $f(A_1 : a_1) = 0$ in the case $f(A_i : a_i) = 1$, $(i = 2, \ldots, n)$ and $f(B : b) = 0$. In the general case in the systems of alternatives can be represented every dependences over attributes meanings (modules of knowledge). The description of the attribute - element of the set $sp(R)$ is the particular case of the knowledge module.

We put the method of alternatives as a basis of the realisation of expert system of diagnostic of tomatoes. In addition the possibilities of structuralisation of the set of attributes and τ-transformation control in the system are called for. The knowledge base of the system contains about 100 modules of the dialog control with user and 600 modules of knowledge about problem domain : 140 modules are descriptions of attributes (alternatives), 200 modules are of the form R_Z , others - of the forms (3), R_Y or alternatives with different attributes. The expert system is realised in the language PASCAL. The experiments with the system demonstrated that the method of alternatives can be used for the solution of practical problems.

EVENING SESSION CONTRIBUTIONS

Proc. IMYCS '86 October 13-17, 1986
Smolenice Castle, CSSR

RECENT RESULTS ON THE THEORY OF HOMOGENEOUS STRUCTURES

Victor Aladyev

SKB MPSM ESSR, Tallinn
Paldiski mnt 171-26
USSR

1. INTRODUCTION

The homogeneous structure(HS) is an information parallel processing system consisting of intercommunicating identical finite automata. although "homogeneous structures" will be the usual term throughout this work, it should be borne in mind that "cellular automata" and so on are essentially synonymous. We can interpret HS as theoretical framework of artificial parallel information processing systems. From the logical point of view the HS is a infinite automaton with characteristic internal structure. The theory of HS can be considered to be the structural and dynamic theory of the infinite automata. HS can serve as the basis for modelling of many discrete processes and they present enough interesting independent objects of investigations as well. During the recent years there has been considerable interest in the theory of HS about which many interesting results have been obtained. Much of this work has been motivated by the growing interest in computer science and biological modelling.

In our previous works [1-5,9,10] we investigated different aspects of the HS theory and their applications in computer science and biological modelling. Results in this directions contributed much that is new to the HS theory and

its applications. However, many questions still remained open in the present topic. In this work we present our recent solutions of a number of open questions in the HS theory. This work is organized so as to discuss the more general problems and results obtained therein. It is rather unfortunate that we have no space here to discuss in detail the basic techniques for solving problems. Exhaustive information about these can be found in Aladyev[6-8]. The all general terms, notions and designations are given in item 2 or are well-known enough.All the others are introduced as the necessity arises.

2. GENERAL DEFINITIONS, CONCEPTS AND NOTIONS

The classical d-dimensional HS(d-HS) is an ordered set of four components

$$d\text{-HS} = \langle Z^d, A, \mathcal{T}^{(n)}, X \rangle,$$

where $A=\{0,1,2,\ldots,a-1\}$ is a set called the state alphabet of the individual finite automata in the structure. Z^d is the set of all d-tuples of integers which is used to name the cell, where Z is the set of integers and is called the array. Each cell z in Z^d can be thought of as the name or address of the particular automata which occupies that position in the array. X, called the neighbourhood index of the d-HS, is an n-tuple of distinct d-tuples of integers and is used to define the neighbours of any cell, i.e., those cells from which the cell z will directly receive information. The neighbourhood index X describes the uniform interconnection pattern(template) among the automata in the d-HS($d \geqslant 1$). The first three above-mentioned components of any d-HS, namely, A, Z^d and X, form a homogeneous space. The state of the entire space is called a configuration(CF) of the space and is any mapping $CF: Z^d \longrightarrow A$, null-CF($\bar{0}$) is a mapping $\bar{0}: Z^d \longrightarrow 0$. C_A denotes the set of all CF with respect to Z^d and A, i.e., $C_A=\{CF \mid CF: Z^d \longrightarrow A\}$. Let c(z) be the current state of the machine located at cell z. The support of a CF c is the set of all cells z such that $c(z)\neq 0$, i.e., the support is the nonquiescent part of CF c. CF with finite support are of considerable interest; the set

of all such CF is denoted by $\overline{C}_A$. The set of all infinite CF of d-HS is denoted by $\overset{\infty}{C}_A$; obviously, $\overline{C}_A \cup \overset{\infty}{C}_A = C_A$ and $\overline{C}_A \cap \overset{\infty}{C}_A = \emptyset$.

The operation of the d-HS is specified by a local function $\delta^{(n)}$ which produces the next state of an individual automaton z in terms of the states of the automata which are directly connected to z. In this work we shall be concerned, in general, with a local function $\delta^{(n)}$, which is defined to be a mapping from A^n to A such that $\delta^{(n)}(0^n)$ always equals 0. The d-HS with such local function is called a stable. For the rest, a local function is any mapping $\delta^{(n)}: A^n \longrightarrow A$.

The simultaneous application of a local function $\delta^{(n)}$ to the neighbourhood of every cell of the homogeneous space defines a global function $\tau^{(n)}$ of the current CF c into the next CF c $\tau^{(n)}$. The operation of a d-HS is particularly simle. If $c=c_0$ is an initial CF of the homogeneous space at time t=0, then the CF at time t=m is $c_0\,\tau^{(n)m}$, the result of applying $\tau^{(n)}$ to the homogeneous space m times. Let $\langle c_0 \rangle_{\tau^{(n)}}$ denote the CF-sequence genetated by function $\tau^{(n)}$ from the CF $c_0 \in C_A$. Now we define the nonconstructibility in d-HS($d \geqslant 1$). Questions of nonconstructibility are fundamental problems in the study of the theoretical properties of d-HS.

DEFINITION 1. CF c is nonconstructible(NCF) for function $\tau^{(n)}$ of d-HS($d \geqslant 1$) iff there does not exist CF $c_0 \in C_A$ such that CF $c_0\,\tau^{(n)}$ contains CF c as subconfiguration.

DEFINITION 2. CF $c \in \overline{C}_A$ is called NCF-1 for function $\tau^{(n)}$ of d-HS iff there exists CF $c' \in \overset{\infty}{C}_A$ such that $c'\,\tau^{(n)}=c$ and there does not exist CF $c^* \in \overline{C}_A$ such that $c^*\,\tau^{(n)}=c$.

DEFINITION 3. Two CF $c_1, c_2 \in \overline{C}_A$ form for function $\tau^{(n)}$ a pair of the mutually erasable CF(MEC) iff $c_1\,\tau^{(n)}=c_2\,\tau^{(n)}$.

Each d-HS($d \geqslant 1$) can be assumed as a parallel formal τ_n-grammar with an axiom $c_0 \in \overline{C}_A$(initial CF in d-HS) and productions $\tau^{(n)}$(global function of d-HS). L(τ_n)-language is the set of all words that can be derived from axiom c_0 by means of applications of global function $\tau^{(n)}$.

The general decomposition problem(GDP) of global functions in d-HS($d \geqslant 1$) can be presented as follows: Can any global function $\tau^{(n)}$ of d-HS be presented in the form of compositi-

on of the finite number of more simple global functions $\tau^{(n_i)}$ $(n_i < n;\ i=\overline{1,k})$?

Now we shall discuss the most significant, in our opinion, recent results in the HS theory and their applications. This work we have done over the years 1984–85 and the first quarter of 1986 [6-8].

3. GENERAL RESULTS

Above all, we turn one's eyes again upon the GDP of global functions in d-HS. The GDP was solved by Aladyev [2] with the help of nonconstructibility approach in d-HS. In our works [3,4] the GDP received further decisions on the basis of other interesting approaches, in the first place, with the help of Shannon's function and on the basis of results in the K-valued logics ($K \geqslant 2$). On a level with well-known GDP it is interesting to investigate the so-called global decomposition problem (GLDP) of global functions of d-HS ($d \geqslant 1$). The GLDP is the question whether or not any global function $\tau^{(n)}$ of d-HS will possess the following representation:

$$\tau^{(n)} = \tau_1^{(n_1)} \tau_2^{(n_2)} \ldots\ldots \tau_k^{(n_k)} \qquad (1)$$

This means that we may use arbitrary global functions as functions $\tau_i^{(n_i)}$ $(i=\overline{1,k})$ in representation (1). Clearly, the positive solution of the GDP for function $\tau^{(n)}$ entail the positive solution of the GLDP for this global function. The inverse assertion is not true, broadly speaking. Therefore, the GDP and the GLDP are not equivalent, generally. In connection with the GLDP Aladyev [6] proved the following results.

THEOREM 1. The GLDP for global functions $\tau^{(n)}$ has negative solution, in general.

THEOREM 2. If for some global function $\tau^{(n)}$ the GDP and the GLDP are equivalent, then for this function these problems are decidable.

The utilization of possibility of representation of local functions $\varphi^{(n)}$ in the form of polynomial in modulo a (a - prime) allow to receive the following interesting result.

THEOREM 3. For any global function $\tau^{(n)}$ in alphabet

$A_p=\{0,1,\ldots,a-1\}$(a - prime) the GDP and the GLDP are equivalent, and algorithmically decidable.

Theorem 3 gives answers on a number of problems from our book[10]. Furthermore, theorems 2 and 3 show that structure of alphabet A of d-HS has of vital importance for the equivalence of the GDP and the GLDP. Using now theorem 3 and proof of theorem 1 the following theorem can be proved.

THEOREM 4. The GDP and the GLDP for function $\tau^{(n)}$ in alphabet A_p have positive solutions iff the function $\tau^{(n)}$ can be presented in the form of composition $\tau^{(n)}=\tau^{(m)}\tau^{(q)}$ $(m,q<n;\ m+q-1=n)$ of two functions in the same alphabet.

From theorem 4 the following interesting result may be drawn.

THEOREM 5. For any integer $n\geqslant 3$ there exist functions $\tau^{(n)}$ in alphabet A_p for which the GDP and the GLDP are equivalent and have negative solutions.

This theorem present just one more proof of negative solutions of the GDP and the GLDP. Using the proof of theorem 5, we can to estimate the quota of functions $\tau^{(n)}$ in alphabet A_p for which the GDP and the GLDP have positive solutions.

THEOREM 6. The GDP and the GLDP for "almost all" functions $\tau^{(n)}$ in alphabet A_p have negative solutions.

Thus, we received slightly unexpected result, namely: quota of all functions $\tau^{(n)}(n\geqslant 3)$ in alphabet A_p, which have positive solutions of the GDP and the GLDP, is equal to zero. From Aladyev's[6] results on the GDP and the GLDP, it can be easily verified that among all functions $\tau^{(n)}(n\geqslant 2)$ in alphabet A_p the infinite hierarchy of complexity with respect to the GDP/GLDP can be established. We shall say that function $\tau^{(n)}$ in alphabet A_p belongs to p-level of complexity(denotion: $\tau^{(n)}\in L(p)$) iff for it there exists representation

$$\tau^{(n)}=\tau_1^{(n_1)}\ldots\ldots\tau_k^{(n_k)}\qquad (n_i\leqslant p<n;(\exists\, i)(n_i=p);i=\overline{1,k})$$

and there does not exist representation of the similar type with $n_i>p(i=\overline{1,k})$. If the GDP(GLDP) for function $\tau^{(n)}$ has negative solution, then $\tau^{(n)}\in L(n)$. Using the above-mentioned results on the GDP and the GLDP, and the proof of theorem 6 we can receive the following correlations:

$$(\forall p \geqslant 2)(\neq\neq L(p) \neq 0) \qquad \lim_{p \to \infty} \neq\neq L(p)/a^{a^p} \geqslant 1 \text{ (a - prime)}$$

From theorem 3 and the definition of complexity with respect to the GDP/GLDP of global functions $\tau^{(n)}$ the following result can be drawn.

THEOREM 7. The problem of determination of p-level of complexity with respect to the GDP/GLDP for arbitrary global function $\tau^{(n)}$ in alphabet A_p is algorithmically decidable.

In view of definition of complexity with respect to the GDP/GLDP of functions $\tau^{(n)}$, Aladyev [6-8] received a number of characteristics of global functions depending on their complexity. From above-mentioned results(theorems 3-7) it is clear that we essentially used the alphabet A_p, since the local function $\varphi^{(n)}$ in this alphabet can be presented in the form of polynomial in modulo a of maximal degree n(a-1) over field A_p, and vice versa. In the case of composite integer a far from each function $\varphi^{(n)}$ in alphabet A can be presented in the polynomial form, generally speaking.

THEOREM 8. For each alphabet $A=\{0,1,\ldots,a-1\}$(a - composite integer) the quota ρ of local functions in the alphabet A, which are presented in the form of polynomial in modulo a, satisfy the following correlation

$$1/a^{a^n-4^n} \leqslant \rho \leqslant 1/a^{a^n-(a-2)^n}$$

Theorem 8 shows that for composite integers a "almost all" local functions $\varphi^{(n)}$ in alphabet A cannot be presented in the form of polynomial in modulo a for enough large integers n or/and a. Aladyev [10] formulated the following problem: Is it possible to define the algebraical system, which permit the polynomial representation of local functions for case of composite integer a, like of the case of prime a. Various algebraical systems have been proposed to answer this question. We present now an algebraical system in which "almost all" local functions in alphabet A(a - composite integer) has representation in the form of polynomial in modulo a. We define the system in the following way. Let on the set $A=\{0,1,\ldots,a-1\}$ (a - composite integer) the usual operation (+) of addition

in modulo a is defined. At the same time, on the set A the binary operation of $\textcircled{x}$-multiplication is introduced in conformity with the following table

$\textcircled{x}$	0	1	2	3	4	5	.....	(a-1)
0	0	0	0	0	0	0	.....	0
1	0	1	2	3	4	5	.....	(a-1)
2	0	2	3	4	5	6	.....	1
3	0	3	4	5	6	7	.....	12
4	0	4	5	6	7	8	.....	123
5	0	5	6	7	8	9	.....	1234
:	:	:	:	:	:	:	:	:
(a-1)	0	(a-1)	...	4	.....			(a-3)(a-2)

(2)

It can be easily seen that operation $\textcircled{x}$-multiplication on the set $A\setminus\{0\}$ form the finite cyclic group $A^{\textcircled{x}}$ of degree (a-1). In view of our above-mentioned suppositions the following general result can be established.

THEOREM 9. There exist an algebraical system $\langle A; +; \textcircled{x}\rangle$ in which "almost each" local function $\varphi^{(n)}$ in the alphabet A can be unequivocally presented in the form of polynomical $P_{\textcircled{x}}(n) \pmod a$, where:

1. (+) is operation of addition in modulo a, which form on the set A the finite additive cyclic group of degree a;
2. ($\textcircled{x}$) is operation of $\textcircled{x}$-multiplication, which is determined by table (2) and which form on the set $A\setminus\{0\}$ the finite cyclic group of degree (a-1);
3. polynomial $P_{\textcircled{x}}(n)= \sum_{i}^{a^n-1} c_i \textcircled{x} X_1^{k_{i_1}} \textcircled{x} X_2^{k_{i_2}} \textcircled{x} \dots \textcircled{x} X_n^{k_{i_n}} \pmod a$ contains no binomials of the form $P_k X_j^k + B_k X_j^{a-k-1}$ (3)

$(0 \leqslant k_{i_j} \leqslant a-1;\ \sum_{j=1}^{n} k_{i_j} \geqslant 1;\ X_j, c_i \in A;\ j=\overline{1,n};\ i=1,a^n-1;$

$P_k+B_k=a;\ P_k, B_k \geqslant 1;\ k=1, \overline{[(a-2)/2]}).$

Theorem 9 plays a very important role in investigations of dynamic properties of d-HS($d \geqslant 1$) in the case of alphabet $A=\{0,1,\dots,a-1\}$(a - composite integer). Furthermore, the theorem gives comfortable analytical representation of functions of a-valued logics in the case of composite integer a. To our knowledge this result is the best of its kind. Using now the proofs of theorems 2 and 3, and the result of theorem 9, it is easily to receive the following theorem.

THEOREM 10. The GDP and the GLDP with respect to the set

of "almost all" global functions $\tau^{(n)}$ in alphabet $A=\{0,1,2,..,a-1\}$(a - composite integer), whose local functions $\delta^{(n)}$ has polynomial representation in the form (3), are equivalent and decidable.

Thus, having a number of results on the problem of decidability of the GDP/GLDP, we cannot spread this achievement on the general case of d-HS, so far. The further investigation on the GDP would be extremely desirable.

The question of the investigation of algorithmical properties of global maps $\tau^{(n)}: C_A^{\infty} \to C_A^{\infty}$ for d-HS($d \geqslant 1$) presents considerable theoretical interest. In connection with this theme the following question arises: Is it decidable whether an arbitrary global map $\tau^{(n)}: C_A^{\infty} \to C_A^{\infty}$ is closed(Closed problem)? For 1-HS Aladyev[10] received the positive answer on this question. This result can be spread on the case d-HS($d \geqslant 2$).

THEOREM 11. The closed problem for d-dimensional($d \geqslant 1$) global maps $\tau^{(n)}: C_A^{\infty} \to C_A^{\infty}$ is decidable.

Aladyev and others[1,10] investigated the problem of interconnection of the minimal size of NCF and MEC in d-HS. However, no one has been able, as yet, to receive a satisfactory solution of this problem. The following result elucidate the reason of such phenomenon.

THEOREM 12. It is impossibly, in general, to receive a satisfactory numerical estimation of the minimal size of NCF in d-HS($d \geqslant 1$) depending on the minimal size of MEC, and vice versa.

This result explain the failure of all previous endeavours on this direction. At the same time we receive the answer on Aladyev's problem 5[10] about the dependence between the minimal size of NCF and MEC in d-HS($d \geqslant 1$). The class of d-HS which has universal reproducing capability in the Moore's sense is enough exceptional in many respect. The next theorem to a certain extent define such class of d-HS($d \geqslant 1$).

THEOREM 13. If d-HS($d \geqslant 1$) possesses the universal reproduction in the Moore's sense then for it there exist NCF-1 without NCF. The inverse assertion is false, in general.

On the basis of theorem 13 can be solved the following

extremely interesting problem: Can a d-HS($d \geqslant 1$) double any finite CF $c \in \overline{C}_A$? The next result gives answer for case d-HS.

THEOREM 14. There exists no d-HS with alphabet A which can double the arbitrary d-dimensional CF $c \in \overline{C}_A$ ($d \geqslant 1$).

In Aladyev[2] the following problem was formulated: Is it decidable whether an arbitrary infinite set $GS \subset \overline{C}_A$ is an $L(\widehat{\tau}_n)$-language? The decisive algorithm is called constructive if it in the case of positive answer give $\widehat{\tau}_n$-grammars themselves which generate $L(\widehat{\tau}_n)$-language GS. In the light of this definition we present now the solution of the more common problem, actually.

THEOREM 15. There exists no the constructive algorithm for solution of the problem: Is it decidable whether an arbitrary infinite set $GS \subset \overline{C}_A$ is an $L(\widehat{\tau}_n)$-language.

In the process of investigation of the GDP by the group methods, Aladyev[10] proved that a semigroup L(a,d) of all d-dimensional maps $\widehat{\tau}^{(n)}: C_A \longrightarrow C_A$ can be presented in the form of union of four subsemigroups, which has no finite systems of generators, and a maximum group G(d). At the same place we formulated the Hypothesis 2: G(d) is a single group, i.e. it consists of global functions $\widehat{\tau}^{(n)}$ which carry out identical maps $\widehat{\tau}^{(n)}: C_A \longrightarrow C_A$, only. The further investigations show that question with group G(d) is open to a certain extent up to this point. We attempted the detailed investigation of binary 1-HS with the purpose of discovering of an one-one maps $\widehat{\tau}^{(n)}: C_A \longrightarrow C_A$, which differ from identical ones. The attempted investigation proved to be a success. The next theorem present the best received result in this direction.

THEOREM 16. For any integer $n \geqslant 3$ there exist at any rate $2^{n-1}-n$ binary 1-dimensional functions $\widehat{\tau}^{(n)}$, which possess the following properties, simultaneously:

1. $\widehat{\tau}^{(n)}$ has no NCF and NCF-1;
2. each CF $c \in \overline{C}_A$ is periodical for such global functions;
3. map $\widehat{\tau}^{(n)}: \overline{C}_A \longrightarrow \overline{C}_A$ is not one-one mapping;
4. for function $\widehat{\tau}^{(n)}$ the GDP has negative solution.

This theorem is essential generalization of lemmas 7,9 from Aladyev[10] but it give not exhaustive solution of the

problem for the case of binary 1-dimensional global functions, even. Whereas, for the case of non-binary maps $\tau^{(n)}$ our hypothesis 2[10] to be wrong, i.e. group G(d) contains nontrivial identical one-one maps. This affirmation is based on the following result.

THEOREM 17. A semigroup L(a,1)($a \geqslant 3$) of all 1-dimensional maps $\tau^{(n)}: C_A \to C_A$ can be presented in the form of union of four subsemigroups, which has no finite systems of generators, and a maximum group G(1), which is union of subgroup T of all identical maps $\tau_o^{(n)}$ ($n > 2$), and symmetrical subgroup P(a) of periodical maps(functions) $\tau^{(n)}$ ($n \geqslant 2$) with the finite system P(a,2) of generators and correlation $\tau^{(n)(a-1)!} = \tau_o^{(2)}$, and, possibly, subgroup of one-one maps, which differ from above-mentioned ones.

Theorem 17 shows that further work on this problem is badly needed. The complexity is one of the most intriguing and vague concepts in most cases. At present we know three approaches to the definition of complexity of the finite objects: combinatorical, probabilistic and algorithmical. For the last case N. Kolmogorov defined the relative complexity of some object G(comparatively of object S) by the minimum length of Turing machine's program of deriving of G from S. Our approach can be also called algorithmical but it differs from Kolmogorov's one[2,3,10]. The essence of our concept of complexity consists in the estimation of complexity of growing of arbitrary finite CF from some primitive CF c_p by means of the finite number of global functions from some set T_f. On the basis of introduced concept of complexity A(X) of the finite CF we presented solutions of a number of problems in the HS theory. The relation between the concept of complexity A(X) and the GDP in d-HS was stated. Furthermore, the relation between A(X) and other famous measures of complexity was presented. However, it is known that our concept of complexity is based on the Hypothesis 3[2]. In Aladyev[6] the proof of this hypothesis was presented. The result is expressed by the following theorem.

THEOREM 18. For any finite alphabet A there exist no the

finite sets of CF $c_i \in \overline{C}_A$ and global functions in alphabet A such that

$$\bigcup_i \langle c_i \rangle_{\tau(n_i)} = \overline{C}_A \qquad (i=\overline{1,k})$$

Theorem 18 allows to give the clean mathematical reasons to a number of results, which were presented in our previous works. On the basis of theorem 18 and the concept of complexity of the finite CF in d-HS($d \geqslant 1$) a number of interesting results can be proved.

THEOREM 19. Supplement of the finite set of $L(\tau_n)$-languages cannot be the language of the same type.

This theorem proves the truth of our hypothesis 4[2], also. Aladyev[2] proved that for d-HS without NCF, but with the set W of NCF-1 there exists no the finite set of CF $c_i \in \overline{C}_A$ such that $\bigcup_i \langle c_i \rangle_{\tau(n)} = \overline{C}_A \setminus W$. Now we shall present essentially more general and very strong result, which gives answer on a number of questions formulated in our previous works[1-5,10].

THEOREM 20. Let $\tau^{(n)}$ be an arbitrary global function in alphabet A (a - prime), which has the set W of NCF and, possibly, NCF-1. Then there exists no the finite set of CF $c_i \in \overline{C}_A$ and global functions $\tau^{(n_i)}$ in alphabet A such that

$$\bigcup_i \langle c_i \rangle_{\tau(n_i)} = \overline{C}_A \setminus W \qquad \text{or} \qquad \bigcup_i \langle c_i \rangle_{\tau(n_i)} = W \qquad (i=\overline{1,k})$$

For the case of composite integer a take place the second correlation.

From this theorem we have a very interesting consequence: sets $\overline{C}_A \setminus W$ and W (W is a set of NCF and, possibly, NCF-1) in the case of prime a cannot be generated by means of the finite sets of CF $c_i \in \overline{C}_A$ and global functions $\tau^{(n_i)}$ $(i=\overline{1,k})$ in alphabet A regardless of global function $\tau^{(n)}$ respect to which the nonconstructibility is considered. Thus, each set of nonconstructible CF (NCF or NCF-1) with respect to the completeness problem possesses the same immunity with the set $\overline{C}_A$.

In our monograph[2] in connection with the investigation of complexity problem of the finite CF in d-HS the following question was formulated: Can the set of CF of each level of complexity be finite? The next theorem to a certain extent clarifies the gist of the matter.

THEOREM 21. There exists the infinite number of basic sets T_f of global functions $\tau^{(n_i)}(i=\overline{1,k})$ with respect to which there exist the infinite sets of the finite CF of the same complexity.

This theorem gives answers on a number of questions presented in Aladyev [2,10]. However, for the complete solution it is necessary in detail to investigate global functions, which form the minimal basic set T_f. We have defined the minimal basic set as a set contained a very insignificant number of global functions. In this direction we have a number of the interesting results.

THEOREM 22. There exists a minimal basic set T_f which contains only four 1-dimensional binary global functions. At any rate a function $\tau^{(n)}$ from the set T_f possesses NCF-1, to say the least.

THEOREM 23. With respect to the minimal basic set T_f of 1-dimensional binary global functions, there exist infinite sets of the finite CF of the same complexity.

THEOREM 24. There exist the minimal basic sets T_f of the binary global functions with respect to which take place the infinite sets of binary functions $\tau^{(n_i)}$ and binary CF $c_i \in \overline{C}_A$ such that sequences $\langle c_i \rangle_{\tau(n_i)}$ contain the binary CF of any given complexity. There exists no the finite basic set T_f of binary global functions with respect to which each sequence $\langle c_o \rangle_{\tau(n)} (c_o \in \overline{C}_A)$ contains binary CF of the limited complexity, only.

Theorem 24 gives answer both on our question [10] and forms the basis of the following extremely interesting result. Above we have noted the difference between concepts of complexity A(X) and K(X)(according to Kolmogorov) of the finite objects. The next theorem establishes the difference between the concepts K(X) and A(X).

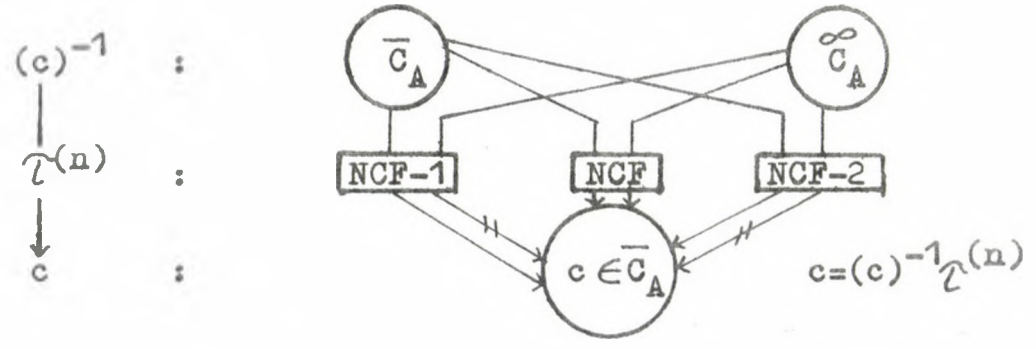
THEOREM 25. There exists the difference of principle with respect to the concepts of complexity K(X) and A(X) between the generative possibilities of the infinite automata MT and 1-HS, which form a base for the above-mentioned concepts of complexity of the finite objects.

This theorem allows to elucidate the difference between of a number of Kolmogorov's and our results on the complexity of the finite objects. We [6] essentially used for the proofs of theorems 20-24 the concept of the minimal basic set T_f and some properties of global functions of T_f; ibid the detailed properties of such minimal basic sets T_f were presented.

Up to now, we considered two concepts of nonconstructibility in d-HS(NCF and NCF-1), only. With the purpose to embrace all possibilities in the problem, we introduced new type of nonconstructibility (NCF-2) in d-HS($d \geqslant 1$) [6].

DEFINITION 4. CF $c \in \overline{C}_A$ is called NCF-2 for function $\tau^{(n)}$ iff there does not exist CF $\overset{\infty}{c} \in \overset{\infty}{C}_A$ such that $\overset{\infty}{c}\,\tau^{(n)}=c$ and there exists CF $c' \in \overline{C}_A$ such that $c'\,\tau^{(n)}=c$.

It is easy to verify that such nonconstructible CF there exist for d-HS($d \geqslant 1$). The next diagram illustrates the essence of all three typies of nonconstructibility in d-HS($d \geqslant 1$).



The interconnection of all typies of nonconstructibility in d-HS expresses the following general result.

THEOREM 26. Each d-HS($d \geqslant 1$) simultaneously has typies of nonconstructibility according to the following table

n/n	NCF	NCF-1	NCF-2	Possibility
1	+	+	+	there exists
2	+	+	−	-//-
3	+	−	+	-//-
4	−	+	+	is absent
5	+	−	−	there exists
6	−	+	−	-//-
7	−	−	+	-//-
8	−	−	−	is absent

The nonempty sets of NCF, NCF-1 and NCF-2 in d-HS($d \geqslant 1$) is infinite, always.

The following theorem gives a criterion of the existence of NCF-2 in 1-HS without NCF.

THEOREM 27. 1-dimensional global function $\tau^{(n)}$ without NCF has a NCF-2 iff the corresponding map $\tau^{(n)}: C_A^{\infty} \to C_A^{\infty}$ is closed.

This criterion is opposite, in a way, to our criterion of the existence of NCF-1 in 1-HS without NCF. From criteria of the existence of NCF-1 and NCF-2 in 1-HS without NCF the following result can be easily received.

THEOREM 28. If 1-dimensional mapping $\tau^{(n)}: C_A^{\infty} \to C_A^{\infty}$ is closed(is not closed) then the corresponding global function $\tau^{(n)}$ without NCF possesses NCF-2(NCF-1).

From theorem 26 and algorithmical decidability of the problems of the existence of NCF and NCF-1 in 1-HS the following theorem can be proved.

THEOREM 29. The problem of the existence of an arbitrary set of NCF, NCF-1 and NCF-2 in 1-HS is decidable.

It is hardly too much to say, that detailed investigation of the concept of mutually erasable CF(MEC) in d-HS present undoubted interest. This investigations will allow to clarify many dynamic properties of d-HS. Similar work we began in our previous books [1,2,10]; now we introduce the new concept of erasability in d-HS.

DEFINITION 5. Two CF $c_1, c_2 \in C_A$ form for function $\tau^{(n)}$ a pair of the MEC-1 iff $c_1 \tau^{(n)} = c_2 \tau^{(n)} = c \in \overline{C}_A$.

The given generalization of the concept of erasability is directly linked with the nonconstructibility problem in d-HS. In view of definition 5 the following result can be proved.

THEOREM 30. 1-dimensional global function $\tau^{(n)}$ possesses NCF or/and NCF-1 iff for it there exists at least a pair of MEC-1.

This result is the essential generalization of the well-known Moore-Myhill's criterion of the existence of NCF in the 1-HS. The next theorem presents a kind of upper boundary for the existence of typies of nonconstructibility in d-HS($d \geqslant 1$).

THEOREM 31. Let NCF0, NCF1, NCF2 be sets of all NCF,NCF-1 and NCF-2 with respect to some global function $\tau^{(n)}$, accor-

dingly. Then for each d-dimensional($d \geqslant 1$) global function $\tau^{(n)}$ take place the following correlations: $NCF0 \subset \overline{C}_A$, $NCF1 \subset \overline{C}_A$ and $NCF0 \cup NCF1 \subset \overline{C}_A$. There exist global functions for which take place the correlation $NCF2 = \overline{C}_A$.

This result gives one of argument in favour of the essential difference between typies of nonconstructibility NCF and NCF-1, on the one hand, and NCF-2, on the other hand. Using now the concept of NCF-2 and proofs of theorems 20 and 26, we can to generalize the theorem 20 on the case of NCF-2.

THEOREM 32. Let $\tau^{(n)}$ be an arbitrary global function in alphabet A(a - prime) having set G of NCF-2. Then there does not exist set of CF $c_i \in \overline{C}_A$ and global functions $\tau_i^{(n_i)}$ in the same alphabet such that

$$\bigcup_i \langle c_i \rangle_{\tau_i^{(n_i)}} = G \qquad (i=\overline{1,k})$$

On the basis of new results on nonconstructibility in our work[6] the following theorem may be drawn.

THEOREM 33. d-dimensional($d \geqslant 1$) global function $\tau^{(n)}$ without NCF possesses NCF-1 iff the corresponding mapping $\tau^{(n)}: \mathfrak{C}_A \to \mathfrak{C}_A$ is not closed, i.e. there exists CF $\mathfrak{c} \in \mathfrak{C}_A$ such that $\mathfrak{c}\,\tau^{(n)} = \overline{0}$.

This theorem gives answer both on a number of questions from our book[2] and our problem 1[10]. Furthermore, it can be used for generalization of a number of the above-mentioned results on nonconstructibility in d-HS($d \geqslant 1$).

THEOREM 34. d-dimensional($d \geqslant 1$) global function $\tau^{(n)}$ without NCF possesses NCF-2 iff the corresponding mapping $\tau^{(n)}: \mathfrak{C}_A \to \mathfrak{C}_A$ is closed; if mapping $\tau^{(n)}: \mathfrak{C}_A \to \mathfrak{C}_A$ is closed (is not closed) then global function $\tau^{(n)}$ without NCF possesses NCF-2(NCF-1). d- dimensional($d \geqslant 1$) function $\tau^{(n)}$ possesses NCF or/and NCF-1 iff for it there exist MEC-1. The problems of the existence of NCF-1 and NCF-2 in d-HS($d \geqslant 1$) without NCF are decidable. If for d-dimensional($d \geqslant 1$) global functuon $\tau^{(n)}$ there does not exist MEC-1 then for it there exist NCF-2; the inverse affirmation to be wrong, in general.

Theorem 34 essentially generalizes the well-known Moore-Myhill's criterion of the existence of NCF in d-HS($d \geqslant 1$).

At the end of the paper we shall present solutions of a number of well-known mathematical problems. These problems once again corroborate the effectiveness of methods of the HS theory for the investigations of the mathematical problems. In the well-known journal "Scientific American" for March 1984 by Haies was presented the unsolved problem "Flights and falls of numbers-hailstones", the essence of which can be formulated as follows.

Let $p_0=n$ be initial number, where $n>0$ is arbitrary integer. The subsequent integers are generated in the following recurrent rule:

$$p_i = \begin{cases} p_{i-1}/2 & \text{, if } p_{i-1} \text{ is even number} \\ 3p_{i-1} & \text{, if } p_{i-1} \text{ is odd number} \end{cases}$$

$$p_0=n \quad (i=1,2,3,\ldots)$$

Numbers p_i form the numerical sequence $SG(p_0)=\{p_i\}$ $(i=0,1,\ldots)$. The following general question can be formulated: Is it possible to state the algorithm of behaviour of $SG(p_0)$-sequence elements(numbers-hailstones) for each integer $p_0>0$?

In this connection we have recently investigated this problem combining some theoretical methods and numerical experiments on the personal computer ISKRA 226[8]. Such approach allows to establish the behaviour of $SG(p_0)$-sequence for any initial integer $p_0>0$. In brief outline the essence of such approach comes to the following.

For numerical experiments with sequence $SG(p_0)$ was worked out mathematical program for personal computer(PC) ISKRA 226 in BASIC-language. This program essentially use some quick parallel algorithms of 1-HS. As a result the computation time of the numerical experiments with $SG(p_0)$-sequences decreases to a large degree. The numerical experiments on the PC ISKRA allow to prove that any $SG(p_0)$-sequence contains element $p_i=4$ $(p_0 \leqslant 2000000;\ i=i(p_0))$. On the other hand, theoretical methods allow to prove that any $SG(p_0)$-sequence contains some element $p_k \leqslant 2000000$. On the basis of the above-mentioned results the following interesting theorem can be formulated.

THEOREM 35. For any integer $p_0>0$ there exists integer

$m=m(p_0)>0$ such that element $m=m(p_0)$ in the $SG(p_0)$-sequence is equal to 4, i.e. each $SG(p_0)$-sequence is periodical with period $l=3$, leading with element $m=m(p_0)$ (integer $p_0>0$).

This theorem gives complete answer on the above-mentioned question. To our knowledge this result is the best of its kind.

S. Ulam [1] has attempted to define heuristic studies of growth in 1-dimensional case on the basis of so-called "unique sum sequences"(USS). Unfortunately, even here it is not easy to establish properties of these USS. By Aladyev and others [1,2,10] theoretical and experimental investigations of a modification of the USS were fulfilled. We investigated such 1-dimensional model of growth by means of 2-HS and computer modelling. Then, Aladyev [8] worked out self-organizing program in BASIC-language for PC ISKRA 226(WANG 2200), which allows to carry out enough wide experiments with the USS.

Let N be set of all positive integers. Define binary operation $w: X+Y \rightarrow P$ $(X,Y,P \in N)$ on the set N. Elements P form a set $N' \subset N$. We shall consider only two typies of binary operation w defined on the set N:

(1) w_1: starting with the integers a and $b(a<b)$ we construct new ones in sequence by considering sums of two previously defined integers, but not including in our collection those integers which can be obtained as a sum of previous ones in more than one way(obligatory conditions: we never add an integer to itself; in addition take part the very right elements of the previous sections of the USS). Such sequences are denoted by USS1(a,b);

(2) w_2: By analogy with w_1, but on condition that the second obligatory condition is omitted. Such sequences are denoted by USS2(a,b).

The elements a_i and a_{i+1} of the USSk(a,b)(k=1,2) are called the <u>twins</u> if $a_{i+1}-a_i=p(a,b)$. The set of all twins for the USSk(a,b)(k=1,2) is denoted by B(p). Both modifications of the USS(a,b) has a number of interesting interpretations. In this direction the following interesting result can be formulated.

THEOREM 36. Each USS2(a,b) has the infinite set of twins at least a type B(a), B(b) or B(a+b). The USS1(a,2a) has the infinite set B(2a); the density of USS1(a,2a) with respect to set N is equal to 0; for USS1(a,2a) there exists formula $a_k=f(k,a)$ which allows to expresses elements a_k of sequence through variables a and k. The USS1(1,b) has the infinite sets B(b), B(b+1) for each integer $b\geqslant 3$; elements a_k of such sequences are expressed by formulas $a_k=f_3(k,b)(b=3)$, $a_k=f_4(k,b)$ $(b=4)$ and $a_k=f_5(k,b)(b\geqslant 5)$. If $a>1$ and $b/a-[b/a]>0$ then in the USS1(a,b) all elements a_k of sequence are expressed by the formula $a_k=b+(k-2)a$ $(k=3,4,5,\ldots)$; the set B(b) in such sequences USS1(a,b) is infinite.

To our knowledge this result is the best of its kind.

At last, we shall present a solution of well-known Steinhays's problem which can be formulated as follows. Let $c_t=$ $=p(1,1)\ldots p(1,t)$ be the first string of length t of binary elements $p(1,i)(i=\overline{1,t})$ and $t\in\{3+4k \text{ or } 4+4k \mid k=0,1,2,\ldots\}$. The elements of the k-th string of length t-k+1 are derived in connection with the following recurrent rule:

$$p(k,i)=p(k-1,i)+p(k-1,i+1)+1 \pmod 2$$
$$(i=\overline{1,t-k+1};\ k=\overline{2,t})$$

As a result, we have a triangular figure F_t which consists of symbols 0 and 1. The string c_t is called a solution of Steinhays's problem for the value t(S(t)-problem) if from it can be derived the figure F_t which contains the same number $t(t+1)/4$ of symbols 0 and 1. We start from some remarks and definitions to present S(t)-problem's solutions.

Let S(t) be the set of all kinds solutions of S(t)-problem. It is easily verified that $S(3)=\{000,011,101,110\}$ and $S(4)=\{0011,0101,1010,1011,1100,1101\}$; these two sets are called basic sets. Solution S(t) is called derivative (D(t)), if it can be presented in the form of concatenation $S(t)=S(t_1)\ldots S(t_n)$ of solutions $S(t_i)$ with $t_i<t$, $\sum t_i=t(i=\overline{1,n})$. A derivative solution D(t) is called basic (B(t)) if in its D(t)-representation $S(t_i)\in S(3)\cup S(4)$ $(i=\overline{1,n})$.

For the purpose of modelling of the process of generation of the above-mentioned figures F_t, we defined a special 2-HS.

The detailed analysis of such 2-HS, which uses the profound properties of global functions $\tau^{(4)}$, shows that for each permitted value $t \geqslant 3$ S(t)-problem has positive solutions. At the same time, a series of the interesting properties of S(t)-problem's solutions can be drawn. On the basis of such analysis and computer modelling on the personal computer ISKRA 226 Aladyev[6-8] proved the following general result.

THEOREM 37. Let S(t), D(t) and B(t) be the sets of all solutions, derivative and basic solutions of S(t)-problem, accordingly. Then for any permitted value $t \geqslant 11$ take place the following correlations:

$$\neq\neq S(t) > \neq\neq D(t) > \neq\neq B(t) > t.$$

For any permitted value t take place the following correlations:

$\neq\neq S(t) \gg 2^{t-r(t)}$, where $r(t) \leqslant [t/2]$, and

$$\neq\neq B(t) \geqslant \begin{cases} 2^{3k-2}, & \text{if } t \in \{3+4k\} \\ 2^{3k}, & \text{if } t \in \{4+4k\} \end{cases} \quad (k=1,2,3,\ldots)$$

where $\neq\neq U$ denotes the cardinality of the set U. Similar results take place for case of derivative solutions, also.

Thus, theorem 37 gives solution of the S(t)-problem formulated by Steinhays for mathematicians(professicnals and amateurs) more 25 years ago. It is important to observe, too, that S(t)-problem can be generalized and results of theorem 37 can be generalized accordingly. Results in this direction can be found in Aladyev[6-8].

I hope that this work will help to clear up some general aspects of the mathematical theory of HS and its applications as well as giving information about the latest our results to scientists working on this topic of the modern cybernetics.

REFERENCES

1. Aladyev V.: To THeory of Homogeneous Structures. Estonian Academic Press. Tallinn 1972, 259 p.
2. Aladyev V.: Mathematical Theory of Homogeneous Structures

and Their Applications. Valgus Press. Tallinn 1980, 268 p.

3. Aladyev V.: New results in the theory of homogeneous structures. Informatik-Skripten 8, Braunschweig 1984, 3-15.
4. Aladyev V.: A few results in homogeneous structures. Parallel Processing by Cellular Automata. PARCELLA-84.Akademie-Verlag. Berlin 1985, 3-16.
5. Aladyev V.: New results in the theory of homogeneous structures. MTA. Szamitastechn. es autom. kut. intez. tanul., no. 158(1984), 3-14.
6. Aladyev V.: Solutions of a Number of Problems in the Theory of Homogeneous Structures. TR-040684, P/A "Silikaat". Tallinn 1985, 60 p.
7. Aladyev V.: Recent Results on the Theory of Homogeneous Structures. TR-061285, P/A "Silikaat". Tallinn 1985, 30 p.
8. Aladyev V.: Architecture and Software of Personal Computer ISKRA 226. SKB MPSM ESSR. Tallinn 1986, 70 p.
9. Parallel Processing and Parallel Algorithms(Ed. by V. Aladyev). Valgus Press. Tallinn 1981, 298 p.
10. Parallel Processing Systems(Ed. by V. Aladyev). Valgus Press. Tallinn 1983, 370 p.

Proc.IMYCS '86 October 13-17,1986
Smolenice Castle,ČSSR

LINEAR VALENCE GRAMMARS

Marian GHEORGHE
University of Bucharest
Computing Centre
14,Academiei Street
70109,Bucharest,Romania

INTRODUCTION. In Păun [5] are introduced and investigated the valence grammars of type-i, i=0,1,2,3. In this paper we examine the relations between type-3 valence languages and the family of linear languages. Also, the generative capacity of linear valence grammars is investigated and some properties of the corresponding family of languages are presented.

In what follows we shall use the formal language theory terminology from Salomaa [6] . The families of languages in the Chomsky hierarchy are denoted by $\mathcal{L}_i$, i=0,1,2,3. By $\mathcal{L}_{LIN}$ and $\mathcal{L}_{MLIN}$ we specify the families of linear and metalinear languages.

An additive valence grammar of type-i (see Păun [5]) is a construct $G = (V_N, V_T, S, P, v)$ where (V_N, V_T, S, P) is a type-i Chomsky grammar and $v : P \longrightarrow Z$. For a derivation $D : S \overset{r_1}{\Longrightarrow} x_1 \overset{r_2}{\Longrightarrow} x_2 \overset{r_3}{\Longrightarrow} \dots \overset{r_n}{\Longrightarrow} x_n$, we define $v(D)=v(r_1)+v(r_2)+\dots v(r_n)$. The language generated by G is:

$$L(G) = \{ x \in V_T^* \mid D : S \overset{*}{\Longrightarrow} x,\ v(D)=0 \} .$$

In a similar way one defines the multiplicative valence grammars: replace Z by Q_+ (the set of positive rational numbers), define $v(D)=v(r_1)x\ v(r_2)x\dots v(r_n)$ and take as "correct" only the terminal derivations D such that v(D)=1.

The families of type-i additive (multiplicative) valence languages are denoted by $\mathcal{AV}_i$ ($\mathcal{MV}_i$), i=0,1,2,LIN,3.

THE GENERATIVE CAPACITY OF LINEAR VALENCE GRAMMARS. First, we examine the relations between the type-3 valence languages and the family of linear languages.

Lemma 1 The families $\mathcal{AV}_3$, $\mathcal{MV}_3$ are uncomparable with $\mathcal{L}_{LIN}$.

Proof. In Păun [5] it is proved that $\mathcal{AV}_3 \subset \mathcal{MV}_3$. Therefore, it is enough to find languages L_1, L_2 such that $L_1 \in \mathcal{AV}_3 - \mathcal{L}_{LIN}$, $L_2 \in \mathcal{L}_{LIN} - \mathcal{MV}_3$.

a) Let us consider the language generated by the grammar G with the rules: r_1: $S \longrightarrow aS$, r_2: $S \longrightarrow bS$, r_3: $S \longrightarrow cS$, r_4: $S \longrightarrow dS$, r_5: $S \longrightarrow d$, with $v(r_1)=v(r_4)=v(r_5)=1$, $v(r_2)=v(r_3)=-1$.

We denote by L_1 the language $L(G) \cap a^*b^*c^*d^* = \{ a^n b^m c^p d^q \mid n,m,p \geq 0,\ q \geq 1,\ n+q=m+p \ (*) \}$.

a1) Clearly, the language L(G) is in $\mathcal{AV}_3$.

a2) We shall prove that L(G) is not in $\mathcal{L}_{LIN}$ by showing that L_1 is not in $\mathcal{L}_{LIN}$.

Let $G' = (V_N, V_T, S, P)$ be a linear grammar such that $L(G')=L_1$. As L_1 is an infinite language, there is a derivation $D : S \overset{*}{\Longrightarrow} uAv \overset{*}{\Longrightarrow} uxAyv \overset{*}{\Longrightarrow} z$, with $xy \neq \lambda$. From the relation (*) we have $xy \notin \{ a, d \}^*$ and $xy \notin \{ b, c \}^*$ and so, it follows that 1) $x=a^i$, $y=b^j$; 2) $x=a^i$, $y=c^j$; 3) $x=b^i$, $y=c^j$; or 4) $x=c^i$ $y=d^j$; $i,j > 0$.

Now, it can be proved that there is a K, such that any word in L(G') çontains at most K symbols d. Hence $L_1 \neq L(G')$ and $L_1 \notin \mathcal{L}_{LIN}$.

Indeed, let D be a derivation as above with A the first recursive symbol and with x,y in the cases 1) or 2). Then the string uxAyv has the form ua^iAb^jv or ua^iAc^jv. It follows that $v=b^sc^td^q$ or $v=c^id^q$, respectively. We obtain that $q \leq K$, where K is the maximum number of occurrences of symbols a(d) that

can be introduced without recursive derivations.

b) Let L_2 be the language generated by the grammar with the rules $S \longrightarrow bSb$, $S \longrightarrow aSa$, $S \longrightarrow c$.

b1) Obviously, $L_2 \in \mathcal{L}_{LIN}$.

b2) We shall prove that $L_2 \notin \mathcal{MV}_3$.

Let $G = (V_N, V_T, S, P, v)$ be a type-3 valence grammar generating L_2. There exists an infinite set of strings of the form : $z = ba^{k_1}\ldots ba^{k_r}ca^{k_r}b\ldots a^{k_1}b$, with arbitrarily large k_i and r. We say that ba^{k_i} is the i-th left component of z. We put $\ell = \max\{ n \mid A \longrightarrow xB \in P,\ n=|x| \}$. For each recursive symbol A, we consider $A \overset{*}{\Longrightarrow} a^nA$ the shortest derivation from A, with $n > 0$. We denote by $\mathcal{A}$ the set of such derivations ($|\mathcal{A}| < \infty$). Let N be $\max\{ n \mid D : A \overset{*}{\Longrightarrow} a^nA,\ D \in \mathcal{A} \}$.

If we take $k_i > K$, where K is $|P|\cdot\ell + |\mathcal{A}|\cdot N$, then for each component of z there is a derivation from $\mathcal{A}$, $D: A \overset{*}{\Longrightarrow} a^nA, v(D) \neq 1$.

Let us consider a terminal derivation D' of z. As r is large enough, then we can choose $r > K$ and it follows that there are two left components i,j with $i \neq j$ and there are two derivations D_i, D_j that contain a subderivation D as above. D occurs p times in D_i and q times in D_j; $p, q > 0$. Using D, p-1 times in D_i and q+1 times in D_j, we obtain a derivation D" with the same valence as D'. Clearly the terminal string obtained is not in L_2. So, $L_2 \notin \mathcal{MV}_3$. □

Now, we examine the relations between the linear valence languages and the families $\mathcal{L}_{LIN}$, $\mathcal{L}_{MLIN}$, and $\mathcal{L}_2$.

Lemma 2 The family $\mathcal{AV}_{LIN}$ is uncomparable with $\mathcal{L}_{MLIN}$ and $\mathcal{L}_2$.

Proof a) $L_3 = \{ a^nb^nc^n \mid n \geq 1 \} \in \mathcal{AV}_{LIN} - \mathcal{L}_2$. Indeed, L_3 can be generated by the grammar with the rules: r_1: $S \longrightarrow aSc$, r_2: $S \longrightarrow B$, r_3: $B \longrightarrow bB$, r_4: $B \longrightarrow b$ and $v(r_1)=1, v(r_2)=0$,

$v(r_3)=v(r_4)=-1$. As $L_3 \notin \mathcal{L}_2$ then $L_3 \in \mathcal{AV}_3 - \mathcal{L}_2$.

b) Let $L_4=\{ a^{n_1}b^{n_1}a^{n_2}b^{n_2}a^{n_3}b^{n_3}a^{n_4}b^{n_4} \mid n_i \geq 1,\ 1 \leq i \leq 4\}$ be a language. We shall prove that $L_4 \in \mathcal{L}_{MLIN} - \mathcal{AV}_{LIN}$.

b1) The language L_4 is generated by the grammar with the rules: $S \longrightarrow AAAA$, $A \longrightarrow aAb$, $A \longrightarrow b$.

b2) Let $G = (V_N, V_T, S, P, v)$ be a linear valence grammar ganarating L_4. For any string x, where

$$x = a^{n_1}b^{n_1}a^{n_2}b^{n_2}a^{n_3}b^{n_3}a^{n_4}b^{n_4},$$

we shall call $a^{n_i}b^{n_i}$ the i-th component, moreover, we call "position" in x each subword of the form α^i, where $\alpha \in \{a,b\}$. If n_i, $1 \leq i \leq 4$, are large enough, the derivation $S \overset{*}{\Longrightarrow} z$ contains a subderivation $D : T \overset{*}{\Longrightarrow} xTy$, where x,y must be α^i, $\alpha \in \{ a,b\}$, $i > o$. Such a derivation can contribute to at most two positions and therefore at least three derivations D_i: $T_i \overset{*}{\Longrightarrow} x_iT_iy_i$, i=1,2,3, with $v(D_i) \neq o$, are necessary.

Suppose $v(D_1)v(D_3) > o$. Considering a terminal derivation D in which D_1 is used k times and D_3 j times, we can develop the proof as in the proof of lemma 1 from Păun [5]. So, we obtain a derivation D' with the same valence as D but with the terminal string not in L_4. Hence $L_4 \neq L(G)$. □

Lemma 3 The families $\mathcal{MV}_{LIN}$ and $\mathcal{L}_2$ are uncomparable.

Proof a) There is a language $L \in \mathcal{MV}_{LIN} - \mathcal{L}_2$. (Indeed, there is a language $L \in \mathcal{MV}_3$ and $L \notin \mathcal{L}_2$ [5]).

b) Let $L_5=\{ a^{k_1}b^{k_1}\ldots a^{k_r}b^{k_r} \mid r \geq 1,\ k_1,\ldots k_r \geq 1 \}$.

b1) The language L_5 is generated by the grammar with the rules $S \longrightarrow SS$, $S \longrightarrow T$, $T \longrightarrow aTb$, $T \longrightarrow ab$. Then $L_5 \in \mathcal{L}_2$.

b2) We can prove that $L_5 \notin \mathcal{MV}_{LIN}$ using a method similar to that given in the proof of the lemma 1 (point b2). □

Theorem 1 1) The following strict inclusions hold:

i) $\mathcal{L}_{LIN} \subset \mathcal{AV}_{LIN} \subset \mathcal{MV}_{LIN}$,

ii) $\mathcal{AV}_3 \subset \mathcal{AV}_{LIN} \subset \mathcal{AV}_2$,

iii) $\mathcal{MV}_3 \subset \mathcal{MV}_{LIN} \subset \mathcal{MV}_2$.

2) The families in the next pairs are uncomparable : $(\mathcal{AV}_3, \mathcal{L}_{LIN})$, $(\mathcal{MV}_3, \mathcal{L}_{LIN})$, $(\mathcal{MV}_3, \mathcal{L}_{MLIN})$, $(\mathcal{AV}_{LIN}, \mathcal{L}_2)$, $(\mathcal{AV}_{LIN}, \mathcal{L}_{MLIN})$, $(\mathcal{MV}_{LIN}, \mathcal{L}_2)$. □

Open problems Which relations there are between $\mathcal{L}_{MLIN}$ and the families $\mathcal{AV}_3$ and $\mathcal{MV}_{LIN}$?

We denote by $\mathcal{V}_1$ the family of unordered generalized vector languages of type-i (see Cremers and Mayer [1], [2]). Using the idea from Dassow and Păun [3] we obtain the result:

Theorem 2 $\mathcal{AV}_{LIN} \subset \mathcal{V}_{LIN}$ and $\mathcal{MV}_{LIN} = \mathcal{V}_{LIN}$. □

A property of the linear valence languages is given by:

Theorem 3 Let $L \in \mathcal{AV}_{LIN}$, then there are $L_1, L_2 \in \mathcal{AV}_3$ such that:

i) $L \subseteq L_1 L_2$,

ii) For any $x \in L_1$ $(y \in L_2)$ there is $y \in L_2$ $(x \in L_1)$ such that $xy \in L$.

Proof Using the classical way, we obtain this result. □

If we replace "additive" by "multiplicative" the result holds too.

REFERENCES

1. A.B. CREMERS, O. MAYER, On matrix languages. Inform. Control 23(1973) 86-96.
2. A.B. CREMERS, O. MAYER, On vector languages. Proc. of Symp. and Summer-School Math. Found of Computer Sci. High Tatras 1973.
3. J. DASSOW, Gh. PĂUN, Regulated rewriting in formal language theory. Akademie-Verlag, Berlin 1986(in print).
4. O.H. IBARRA, S.K. SAHNI, C.E. KIM, Finite automata with multiplication. Theoretical Computer Sci. 2(1976)271-294.
5. Gh. PĂUN, A new generative device: valence grammars, Rev. Roum. Math. Pures et Appl. 25(1980) 911-924.
6. A. SALOMAA, Formal languages. Academic Press, New York and London 1973.

Proc. IMYCS '86 October 13-17, 1986
Smolenice Castle, CSSR

A GENERALIZATION OF SATURATED GRAPHS FOR FINITE LANGUAGES

Zs. Tuza

Computer and Automation Institute
Hungarian Academy of Sciences
H-1111 Budapest, Kende u. 13-17, Hungary

The study of finite languages is a relatively new field within the theory of formal languages and has not been investigated extensively. Finite structures however, have much importance in the theory of computers and they have gained an increasing interest in the past few years. In a sense, the study of finite languages needs some methods different from those which are developed for classical Chomsky-type languages.

In this note we extend a graph-theoretic concept for finite languages and raise the following problem. Let L_o be a given finite language of length k, i.e. all words of L_o consist of exactly k characters. Denote by $\underline{L}(n,k)$ the set of all languages of length k, over an n-element alphabet. A language $L \in \underline{L}(n,k)$ is called L_o-saturated if $L_o \not\subset L$ but $L_o \subset L'$ for every $L' \in \underline{L}(n,k)$ such that $L \subsetneq L'$.

Problem 1. If n and L_o are given,

(i) determine $sat(n,L_o) = \min \{|L| : L \in \underline{L}(n,k), L \text{ is } L_o\text{-saturated}\}$;

(ii) describe the structure of L_o-saturated languages.

This problem can be raised in a far more general setting. Consider a collection $\underline{S}$ of given structures (graphs directed graphs, hypergraphs, partially ordered sets, finite languages, sets of sequences or permutations, etc.). Let $\underline{P}$ be a property such that if $S,S' \in \underline{S}$ and $S \subset S'$ then $\underline{P}$ holds for S

only if it holds for S'. Let $\mu: \underline{S} \to \mathbb{R}^+$ be an increasing function (i.e., $\mu(S) \leq \mu(S')$ when $S \subset S'$). Call a structure $S \in \underline{S}$ $\underline{P}$-saturated if S does not have property $\underline{P}$ but $\underline{P}$ holds for every $S' \in \underline{S}$ such that $S \subsetneq S'$.

Problem 2. Determine

$$sat(\underline{S},\underline{P},\mu) = \min \{\mu(S) : S \in \underline{S},\ S \text{ is } \underline{P}\text{-saturated}\} .$$

In Problem 1, $\mu(L)$ is the number of words in L, $\underline{P}$ is the property that L contains L_o, and $\underline{S} = \underline{L}(n,k)$.

Though Problem 2 is rather too general, there are several interesting particular cases of it which seem to be worth studying. For example, in Problem 1 we can replace $|L|$ by any complexity measure. These measures usually are defined for grammars but they produce complexity measures for the language in a natural way:

$$prod\ (L) := \min_G\ prod\ (G) \quad ; \quad symb\ (L) := \min_G\ symb\ (G) \quad ;$$

and so on, where the minimum is taken over all grammars G of a given type (regular, context-free, etc.) and producing L.

Another possibility is to consider subclasses of $\underline{L}(n,k)$. The simplest particular case is the family of symmetric languages of length 2: $\underline{L}^*(n) = \{L \in \underline{L}(n) : ab \in L \Leftrightarrow ba \in L\}$, where $\underline{L}(n) = \underline{L}(n,2)$. Define $sat^*(n,L_o) = \min \{|L| : L \in \underline{L}^*(n),\ L \text{ is } L_o\text{-saturated}\}$.

As a matter of fact, the restriction should not necessarily be done for L_o, because the problem of determining $sat^*(n,L_o)$ makes sense even if $L_o \notin \underline{L}^*(n)$. Similarly, in Problem 2 we only have to assume that $\underline{P}$ holds for all structures $S \in \underline{S}$ which are maximal under inclusion (i.e., $S \subset S' \in \underline{S}$ implies $S=S'$). In case of $\underline{L}^*(n)$ however, one can assume that L_o is symmetric, without loss of generality, as shown by the next proposition. For the moment, let us write $L_1 \sim L_2$ if $ab \in L_i$ implies $ab \in L_j$ or $ba \in L_j$ where $j=3-i$ for $i=1,2$.

Proposition 3. If $L_1 \sim L_2$ then $sat^*(n,L_1) = sat^*(n,L_2)$.

Let $L_1 \simeq L_2$ denote that $L_1 \sim L_2$ and $\{ab,ba\} \subset L_1$ if and only if $\{ab,ba\} \subset L_2$.

Problem 4. When does $L_1 \simeq L_2$ imply $sat(n,L_1) = sat(n,L_2)$?

Problem 5. Let $L_1 \simeq L_2$ and suppose that L_1' is L_1-saturated. Under what assumptions does an L_2-saturated L_2' exist such that $L_2' \simeq L_1'$?

If L_o belongs to a subclass of $\underline{L}(n)$ then the function "sat" may or may not be restricted to the same subclass. An interesting case is when L_o is asymmetric, i.e., $ab \in L_o$ implies $ba \notin L_o$. Set $\underline{L}^-(n) = \{L \in \underline{L}(n) : L \text{ is asymmetric}\}$ and define $sat^-(n,L_o) = \min \{|L| : L \in \underline{L}^-(n), L \text{ is } L_o\text{-saturated}\}$. Trivially,

$$sat^- \geq sat \quad \text{and} \quad sat^* \geq sat \qquad (1)$$

for all possible n and L_o.

Problem 6. Find necessary and/or sufficient conditions for having equality in (1).

Connection with graphs and digraphs

If $L \in \underline{L}(n)$, L can be identified with a directed graph $G = G_L$ on n vertices: ab is an edge of G if and only if $ab \in L$. Similarly, there is an obvious one-to-one correspondence between symmetric languages $L \in \underline{L}^*(n)$ and undirected graphs on n vertices. Hence, the next theorem is equivalent to a result of Kászonyi and myself [4].

Theorem 7. For every symmetric language L_o there exists a constant $c = c(L_o)$ such that $sat^*(n,L_o) \leq cn$.

This result can be extended from $\underline{L}^*(n)$ to $\underline{L}(n)$ (the two proofs however, are somewhat different).

Theorem 8. For every language L_o of length 2 there exists a constant $c = c(L_o)$ such that $sat(n,L_o) \leq cn$.

The following conjecture seems to be the most challenging open question at the moment.

Conjecture 9. For every given L_o, the limit $\lim_{n\to\infty} sat(n,L_o)/n$ exists.

For graphs, I raised this question several years ago. I find it worth mentioning in its original form also, because it has not been published previously and an affirmative answer would be fundamental in the theory of saturated graphs.

Conjecture 10. For every graph F there exists a constant $c = c(F)$ such that $sat(n,F) = cn + o(n)$.

In general, this conjecture is still unsolved. Of course, it is known to be true when there is an exact formula for the value $sat(n,F)$. These cases are when F is a complete graph (Erdős, Hajnal and Moon [3]) or F consists of pairwise disjoint edges (by a structure theorem of Mader [5]) or it is a cycle of four vertices (Ollman [6]) or a star or a path (Kászonyi and Tuza [4]).

Quite recently, some development has been achieved concerning Conjecture 10. Truszczyński and I [7] proved it for some class of graphs; more precisely, an infinite class of graphs F has been constructed such that $c(F)$ exists and is smaller than 1. Certainly, if F is connected and $c(F) < 1$ then F must be a tree. This property however, does not necessarily hold for the connected components of an F. There are several disconnected examples F having no more than one tree component. Therefore, as a first step towards the solution of Conjecture 10, it would be interesting to see the answer to the following question.

Problem 11. Characterize those trees T for which

$$c(T) = \lim_{n \to \infty} sat(n,T)/n < 1 .$$

For some particular values of $c(F)$, such a description will be found in [7]. For example, $c(F) = 1/2$ if and only if F has no isolated edges but it contains a connected component isomorphic to a path of 3 or 4 vertices.

We note that in all known cases

$$sat(n,F) = cn + O(1) \qquad (2)$$

holds, in spite of the surprising fact that $sat(n,F)$ is not always an increasing function of n. It may happen that Conjecture 10 (and 9 also) holds in the somewhat stronger form (2). Let us mention one more problem which seems to be important.

Conjecture 12. If L_o has length k, then $sat(n,L_o)=O(n^{k-1})$.

Related questions on graphs

Let F be a simple undirected graph. Finding the value of $sat(n,F)$ is the natural counterpart of the well-known Turán-type problems where the question is to determine the maximum number of edges in graphs of n vertices which do not contain F as a subgraph. (Such a graph is always F-saturated.) This topic has an extended literature and is cleared up almost perfectly when the chromatic number of F is at least 3 (for the details and references, see [2]).

Another fruitful direction is related to the case when the graph F is complete. Then a graph G is saturated if and only if its complement $\overline{G}$ satisfies the following property: The minimum cardinality of a vertex set meeting all edges of $\overline{G}$ decreases whenever an edge is deleted from $\overline{G}$. This "τ-critical" property is important in the study of hypergraphs also (see e.g. [1] or [8]).

Strongly saturated and weakly saturated structures

There is another possibility to formulate Problem 2. Let $\eta : \underline{S} \to \mathbb{R}^+$ be a given increasing function. Call a structure S η-*saturated* if $\eta(S') > \eta(S)$ for all S', $S \subsetneq S' \in \underline{S}$. If η is defined as the characteristic function of $\underline{P}$, i.e. $\eta(S) = 1$ if S satisfies $\underline{P}$ and $\eta(S) = 0$ otherwise, then S is η-saturated if and only is it is $\underline{P}$-saturated. This formulation makes it possible to weaken the definition in two ways.

For the given collection $\underline{S}$ of structures, let $\mu, \eta : \underline{S} \to \mathbb{R}^+$ be two increasing functions. We say that an $S \in \underline{S}$ is *strongly* η-*saturated* if $\eta(S') > \eta(S)$ for all $S' \in \underline{S}$, $S \subsetneq S'$. Define

$$\text{s-sat}(\underline{S}, \eta, \mu) = \min \{ \mu(S) : S \in \underline{S}, S \text{ is strongly } \eta\text{-saturated} \} .$$

(For a given structure S_o, putting $\eta(S)=0$ if $S_o \not\subset S$ and $\eta(S)=1$ if $S_o \subset S$, saturated and strongly saturated structures coincide.)

As another extension, *weakly* η-*saturated* structures can be defined as follows. Call a sequence $S_1, S_2, \ldots, S_k$ *strong chain* if $S_1 \subsetneq S_2 \subsetneq \ldots \subsetneq S_k$, $S_i \in \underline{S}$ for $1 \leq i \leq k$, and for all i $(1 \leq i \leq k-1)$ $S_i \subset S \subset S_{i+1}$ implies $S=S_i$ or $S=S_{i+1}$ if $S \in \underline{S}$. Suppose there is exactly one maximal $S^* \in \underline{S}$. Then an $S \in \underline{S}$ is called weakly η-saturated if there exists a strong chain $S_1, \ldots, S_k$ with $S_1=S$, $S_k=S^*$, such that $\eta(S_i) < \eta(S_{i+1})$ for all i, $1 \leq i \leq k-1$. The problem is to find

$$\text{w-sat}(\underline{S}, \eta, \mu) = \min \{ \mu(S) : S \in \underline{S}, S \text{ is weakly } \eta\text{-saturated} \} .$$

It may happen that $\underline{S}$ contains several maximal members. Then there are two possibilities to extend the definition. Let us call a chain $S_1, \ldots, S_k$ η-*increasing* if $\eta(S_i) < \eta(S_{i+1})$ for all $i \leq k-1$. A structure $S \in \underline{S}$ may be called weakly η-saturated if (a) for every maximal $S^* \in \underline{S}$, $S \subset S^*$, there is an η-increasing strong chain $S_1, \ldots, S_k$ with $S_1=S$ and $S_k=S^*$; or, under an even weaker assumption if (b) there is an η-increasing strong chain $S_1, \ldots, S_k$ such that $S_1=S$ and S_k is a maximal member of $\underline{S}$.

An important particular case of strongly and weakly saturated structures occurs when a given S_o is fixed and $\eta(S)$ denotes the number of substructures of S which are isomorphic to

S_o. In this case our definitions coincide with those given for graphs in [2], and this is the reason why we use the same terminology. Note however, that strongly saturated property is weaker than the saturated one i.e., if $\eta(S)$ is the number of substructures of S, isomorphic to S_o, and $\underline{P}$ means $S_o \subset S$, then

$$w\text{-}sat(\underline{S}, \eta, \mu) \leq s\text{-}sat(\underline{S}, \eta, \mu) \leq sat(\underline{S}, \underline{P}, \mu) . \quad (3)$$

Now it is quite natural to put the following question.

Problem 13. Under which conditions does equality hold in one or both parts of (3)?

This question seems to be very hard to answer, even in the "simple" particular cases. For example, if L_o is a complete and symmetric language of length 2 then equality holds in (3) but to show this statement one has to use linear algebraic methods.

In [9] I proved that every $L \in \underline{L}(n)$ can be generated by $O(n^2/\log n)$ context-free productions, and this upper bound is best possible. So at the end of this paper let me raise a problem which differs from the previous ones.

Problem 14. If L_o and n are given, determine $\{\max\ c(L) : L \in \underline{L}(n)$, L is L_o-saturated$\}$, where c(L) is the minimum number of productions in a context-free grammar generating L.

References

1 C.Berge, "Graphs and Hypergraphs," North-Holland, 1973.
2 B.Bollobás, "Extremal Graph Theory," Academic Press, 1978.
3 P.Erdős, A.Hajnal and J.W.Moon, A problem in graph theory, Amer. Math. Monthly 71 (1964) 1107-1110.
4 L.Kászonyi and Zs.Tuza, Saturated graphs with a minimal number of edges, J. Graph Theory, in print
5 W.Mader, 1-Faktoren in Graphen, Math. Ann. 201 (1973) 269-282.
6 L.T.Ollman, $K_{2,2}$-saturated graphs with a minimal number of edges, in: Proc. 3rd South-East Conference on Combinatorics, Graph Theory and Computing, pp. 367-392.
7 M.Truszczyński and Zs.Tuza, Asymptotic results on saturated graphs, in preparation
8 Zs.Tuza, Critical hypergraphs and intersecting set-pair systems, J. Combinatorial Th., Ser. B, 39 (1985) 134-145.
9 Zs.Tuza, On the context-free production complexity of finite languages, submitted

1985-BEN MEGJELENTEK:

166/1985 Radó Péter: Információs rendszerek számitógépes tervezése

167/1985 Studies in Applied Stochastic Programming I. Szerkesztette: Prékopa András /utánnyomás/

168/1985 Böszörményi László - Kovács László - Martos Balázs - Szabó Miklós: LILIPUTH

169/1985 Horváth Mátyás: Alkatrészgyártási folyamatok automatizált tervezése

170/1985 Márkus Gábor: Algoritmus mátrix alapu logaritmus kiszámitására kriptográfiai alkalmazásokkal

171/1985 Tamás Várady: Integration of free-form surfaces into a volumetric modeller

172/1985 Reviczky János: A számitógépes grafika terület-kitöltő algoritmusai

173/1985 Kacsukné Bruckner Livia: Mozgáspálya generálás bonyolult geometriáju felületek 2 1/2D-s NC megmunkálásához

174/1985 Bolla Marianna: Mátrixok spektrálfelbontásának és szinguláris felbontásának módszerei

175/1985 Hannák László, Radó Péter: Adatmodellek, adatbázis-filozófiák

176/1985 Számitógépes képfeldolgozási és alakfelismerési kutatók találkozója.
Szerkesztette: Csetverikov Dmitirj,
Főglein János és Solt Péter

177/1985 Gyárfás András: Problems from the world surrounding perfect graphs

178/1985 PUBLIKÁCIÓK'84
Szerkesztette: Petróczy Judit

Készült az Országos Széchényi Könyvtár Sokszorosító üzemében, Budapest. Felelős vezető: Rosta Lajosné
Példányszám: 320
Terjedelem: 38 A/5 iv
Munkaszáma: 86 242