Lecture Notes in
Computer Science

Edited by G. Goos and J. Hartmanis

380

J. Csirik J. Demetrovics
F. Gécseg (Eds.)

Fundamentals of Computation Theory

International Conference FCT '89
Szeged, Hungary, August 21–25, 1989
Proceedings

Springer-Verlag
New York Berlin Heidelberg London Paris Tokyo Hong Kong

Editors

János Csirik
Férenc Gécseg
József Attila Tudományegyetem Bolyai Intézete
Aradi vértanúk tere 1, H-6720 Szeged, Hungary

János Demetrovics
Computer and Automation Institute
Hungarian Academy of Sciences
P.O. Box 63, H-1502 Budapest, Hungary

CR Subject Classification (1987): D.2.4, E.5, F, I.1.2, I.2–3

ISBN 3-540-51498-8 Springer-Verlag Berlin Heidelberg New York
ISBN 0-387-51498-8 Springer-Verlag New York Berlin Heidelberg

Printing and binding: Druckhaus Beltz, Hemsbach/Bergstr.
2145/3140-543210 – Printed on acid-free paper

PREFACE

This volume constitutes the proceedings of the conference on Fundamentals of Computation Theory held in Szeged, Hungary, August 21-25, 1989. The conference is the seventh in the series of the FCT conferences initiated in 1977 in Poznan-Kornik, Poland. The conference was organized by the Attila József University (Szeged) with cooperation of the Computer and Automation Institute of the Hungarian Academy of Sciences.

The papers in this volume are the texts of invited addresses and shorter communications falling in one of the following sections:

Efficient Computation by Abstract Devices: Automata, Computability, Probabilistic Computations, Parallel and Distributed Computing

Logics and Meanings of Programs: Algebraic and Categorical Approaches to Semantics, Computational Logic, Logic Programming, Verification, Program Transformations, Functional Programming

Formal Languages: Rewriting Systems, Algebraic Language Theory

Computational Complexity: Analysis and Complexity of Algorithms, Design of Efficient Algorithms, Algorithms and Data Structures, Computational Geometry, Complexity Classes and Hierarchies, Lower Bounds

The shorter communications were selected on March 21 and 22, 1989 at the Program (Selection) Committee Meeting in Szeged from the large number of papers submitted to FCT '89.

Thanks are due to the members of the Program Committee for their work in evaluating the submitted papers, to the members of the Organizing Committee for their hard job in all organizational matters as well as to all referees of FCT '89.

Szeged, August 1989

János Csirik János Demetrovics Ferenc Gécseg

CONFERENCE COMMITTEES

PROGRAM COMMITTEE

G. Ausiello, J. Berstel, L. Budach, R.G. Bukharajev, Phan Dinh Dieu, P. van Emde Boas, the late A.P. Ershov, F. Gécseg, J. Gruska, J. Hartmanis, J. Heintz, G. Hotz, K. Indermark, H. Jürgensen, M. Karpinski, L. Lovász, O.B. Lupanov, G. Mirkowska, A. Mostowski, A. Pultr, J.H. Reif, G. Rozenberg, J. Sakarovitch, A. Salomaa, E. Szemerédi, H. Thiele, I. Wegener, Wu Wen-Tsün

ORGANIZING COMMITTEE

J. Demetrovics (Chairman), J. Csirik (Secretary), Z. Ésik (Secretary), M. Bartha, Gy. Horváth, J. Virágh

CONFERENCE CHAIRMAN

Ferenc Gécseg

REFEREES OF FCT '89

Ablayev, F.M.
Adámek, J.
Albrecht, A.
America, P.H.M.
Arnolds, A.
Asveld, P.R.J.
Badouel, E.
Banachowski, L.
Barendregt, H.P.
Becker, B.
Bilardi, G.
Bloom, S.L.
Boanon, L.
Brandenburg, F.J.
Briot, J.-P.
Chlebus, B.S.
Chrétienne, P.
Chytil, M.
Courcelle, B.
Culik, K. II
Dahlhaus, E.
Darondeau, P.
Dassow, J.
Dauchet, M.
Delporte-Gallet, C.
Diekert, V.
Diks, K.
Ďuriš, P.
Ehrig, P.
Engelfriet, J.
Enikeev, A.I.
Grigorieff, S.
Habel, A.
Haddad, S.
Hansel, G.
Harju, T.
Hoffmann, F.
Hromkovič, J.
Huttenlocher, D.
Jędrzejowicz, J.
Jung, H.
Karhumäki, J.
Kleijn, H.C.M.
Klop, J.W.
Kluźniak, F.
Kolla, R.
Korec, I.
Kreowski, H.-J.
Krob, D.
Kuchen, H.
Kudlek, M.
Kusnetsov, S.E.
Lehmann, T.
Lescanne, P.
Linna, M.
Loeckx, J.
Lukaszewicz, W.
Machi, A.
Marek, I.
Mazoyer, J.
Mehlhorn, K.
Meinel, C.
Ch. Meyer, J.-J.
Molitor, P.
Müller, F.
Nait Abdallah, M.A.
Nasirov, I.R.
Niwiński, D.
Nurmeev, N.N.
Oberschelp, W.
Ollagnier, J.M.
Orlowska, E.
Pelletier, M.
Perrot, J.-F.
Pin, J.E.
Privara, I.
Ranjan, D.
Reutenauer, C.
Rovan, B.
Rudak, L.
Ruohonen, K.
Ružička, P.
Salimov, F.I.
Salwicki, A.
Samitov, R.K.
Schinzel, B.
Schupp, P.E.
Seidl, H.
Sieber, K.
Skowron, A.
Solovjev, V.D.
Spaniol, O.
Stabler, E.P. Jr.
Štěpánek, P.
Štura, J.
Szepietowski, A.
Teitelbaum, T.
Thomas, W.
Treinen, R.
Validov, F.I.
Vitányi, P.M.B.
Vogler, H.
Waack, S.
van Westrhenen, S.C.
Wiedermann, J.
Wolf, G.

Contents

ON WORD EQUATIONS AND MAKANIN'S ALGORITHM

Habib Abdulrab, Jean-Pierre Pécuchet
Laboratoire d'Informatique de Rouen et LITP.
Faculté des Sciences, B.P. 118, 76134 Mont-Saint-Aignan Cedex †
E.m.: mcvax!inria!geocub!abdulrab mcvax!inria!geocub!pecuchet

ABSTRACT. — We give a short survey of major results and algorithms in the field of solving word equations, and describe the central algorithm of Makanin.

Introduction

An algebra equipped with a single associative law is a **semigroup**. It is a **monoid** when it has a unit. The free monoid generated by the set A (also called **alphabet**) is denoted by A^*. Its elements are the **words** written on the alphabet A, the neutral element being the empty word denoted by 1. The operation is the concatenation denoted by juxtaposition of words. The **length** of a word w (the number of letters composing it) is denoted by $|w|$. For a word $w = w_1 \dots w_n$, with $|w| = n$, we denote by $w[i] = w_i$ the letter at the ith position. The number of occurrences of a given letter $a \in A$ in a word w, will be denoted by $|w|_a$.

In this terminology, the term algebra (in the sense of [Fag Hue], [Kir]) built on a set of variables V, a set C of constants, and a set of operators constituted of an associative law, is nothing else than the free monoid $T = (V \bigcup C)^*$ over the alphabet of **letters** $L = V \bigcup C$.

A unifier of two terms $e_1, e_2 \in T$ is a monoid morphism $\alpha : T \longrightarrow T$ (i.e. a mapping satisfying $\alpha(mm') = \alpha(m)\alpha(m')$ and $\alpha(1) = 1$), leaving the constants invariant (i.e. satisfying $\alpha(c) = c$ for every $c \in C$) and satisfying the equality $\alpha(e_1) = \alpha(e_2)$.

The pair of words $e = (e_1, e_2)$ is called an **equation** and the unifier α is a **solution** of this equation.

A solution $\alpha : T \longrightarrow T'$ **divides** a solution $\beta : T \longrightarrow T''$ if there exists a **continuous** morphism $\theta : T' \longrightarrow T''$ (i.e. satisfying $\theta(x) \neq 1$ for every x) such as $\beta = \alpha\theta$. We also say that α is **more general** than β. A solution α is said to be **principal** (or **minimal**) when it is divided by no other but itself (or by an **equivalent** solution, i.e. of the form $\alpha' = \alpha\theta$ with θ, an isomorphism).

The two main problems concerning systems of equations are the existence of a solution, and the computation of the set of minimal solutions (denoted by μCSU_A in [Fag Hue]). All these problems reduce to the case of a single equation, as by [Alb Law] every infinite system of equations is equivalent to one of its finite subsystems, and a finite system can be easily encoded in a single equation [Hme].

† This work was also supported by the Greco de Programmation du CNRS and the PRC Programmation Avancée et Outils pour l'Intelligence Artificielle.

The study of properties and stricture of the set of solutions of a word equation was initiated by Lentin and Schützenberger ([Len Sch], [Len]) in the case of constant-free equations ($C = \emptyset$).

In particular, Lentin shows that every solution is divided by a unique minimal one and gives a procedure (known as the **pig-pug**) allowing to enumerate the set of minimal solutions. This procedure extends without difficulty to the general case of an equation with constants (cf. [Plo], [Pec1]). The minimal solutions are obtained as labels of some paths of a graph. When this graph is finite, as in the case when no variable appears more than twice, we obtain a complete description of all solutions.

The problem of the existence of a solution was first tackled by Hmelevskii who solved it in the case of three variables [Hme], then by Makanin who solved the general case [Mak1]. He gave an algorithm to decide whether a word equation with constants has a solution or not.

This paper is divided into two parts. The first one will be devoted to a brief presentation of the pig-pug method which gives, for simple cases, the most efficient unification algorithm. The rest of this paper will be devoted to Makanin's Algorithm [Mak1] as it is implemented by Abdulrab [Abd1]. In order to keep a reasonable size to this paper, most of the proofs will be omitted.

1. The pig-pug

In the remaining part of this paper, we assume without loss of generality, that the alphabets of variables $V = \{v_1 \dots v_n\}$ and of constants $C = \{c_1 \dots c_m\}$ are finite and disjoint. We make the convention to represent the variables by lower-case letters, as $x, y, z \dots$, and the constants by upper-case letters as $A, B, C \dots$. We call **length** of an equation $e = (e_1, e_2)$ the integer $d = |e_1 e_2|$.

The **projection** of an equation e over a subset Q of V is the equation obtained by "erasing" all the occurrences of $V \setminus Q$. Consequently, an equation has 2^n projections $(\Pi_Q e_1, \Pi_Q e_2)$ where $\Pi_Q : (V \bigcup C)^* \longrightarrow (Q \bigcup C)^*$ is the projection morphism.

One easily proves the following proposition which reduces the research of a solution to that of a continuous one.

Proposition 1.1 *An equation e has a solution iff one of its projections has a continuous solution.* ∎

The pig-pug method consists in searching for a continuous solution α in the following manner: it visits the lists $e_1[1], \dots, e_1[|e_1|]$ and $e_2[1], \dots, e_2[|e_2|]$ of symbols of e from left to right and at the same time, one tries to guess how their images can overlap. At each step, one makes a non deterministic choice for the relative lengths of the images of the first two symbols $e_1[1]$ et $e_2[1]$. According to the choice made :

$$|\alpha(e_1[1])| < |\alpha(e_2[1])|, \quad |\alpha(e_1[1])| = |\alpha(e_2[1])|, \quad |\alpha(e_1[1])| > |\alpha(e_2[1])|$$

one applies to the equation one of the three substitutions to variables :

$$e_2[1] \leftarrow e_1[1]e_2[1], \quad e_2[1] \leftarrow e_1[1], \quad e_1[1] \leftarrow e_2[1]e_1[1].$$

The process is repeated until the **trivial** equation (1,1) is obtained.

The application of the pig-pug method to all the projections of an equation gives a graph labeled by the three previous transformations. The nodes of the graph are the transformed equations.

The following result, a proof of which can be found in [Pecl], shows that this graph enumerates all the minimal solutions.

Theorem 1.2 [Len] *The set of minimal solutions of a word equation is given by the labels of the paths linking the root to the trivial equation in the pig-pug graph.* ▮

One can consider the graph associated with an equation e, as a deterministic automaton M, in which :

1) the alphabet is given by the projections and the variable substitution.
2) the states of the automaton are the vertices of the graph.
3) the final state is $(1, 1)$.

The language accepted by this automaton, designated $L(M)$, is precisely the set of all the minimal solutions of e, that is, a complete set of minimal unifiers.

When the graph is finite, $L(M)$ is a regular language. But in the general case the automaton is infinite. And $L(M)$ in not necessarily regular, and not even a context-free language, (for example, this is the case of the automaton M associated with the equation $e = (xAAyx, AxxBz)$).

Note that $L(M)$ is not necessarily minimal, in the sense that, equivalent minimal unifiers can be generated within $L(M)$.

In the general case, the pig-pug's graph will be infinite. However one can always decide the existence of a solution by:

Theorem 1.3 [Mak2] *One can construct a recursive function F such that, if an equation of length d has a solution, then there exists one in which the length of the components of the solutions are bounded by $F(d)$.* ▮

The only known function F is that derived from Makanin's algorithm that we will see now. Another reason for the study of this algorithm is that it leads to a better pruning of the graph, and is more efficient than the pig-pug method in some cases.

2. Length equations

Before describing Makanin's algorithm, we introduce the notion of length equations which is related to integer programming.

First note that if $Card(C) = 0$, the equation e has necessarily the trivial solution $\alpha(v) = 1$, for every $v \in V$. Consequently, we can assume that $Card(C) > 0$. An equation is called **simple** if $card(C) = 1$. Such equations are related to integer equations in the following way:

Let e be a simple equation, consider the commutative image e' of e:

$$(v_1^{p_1} \dots v_n^{p_n} c_1^{p_{n+1}}, v_1^{q_1} \dots v_n^{q_n} c_1^{q_{n+1}})$$

The linear diophantine equation: $(p_1-q_1)v_1'+\ldots+(p_n-q_n)v_n' = (q_{n+1}-p_{n+1})$. is called the **length equation** associated with e.

The isomorphism between $c_1{}^*$ and $(N,+)$ gives the following correspondence:

Proposition 2.1 *There is a bijective correspondence between the solutions of a simple equation and the non-negative integer solutions of its length equation.* ▮

With every equation e we associate the simple equation e' obtained by the substitution of all the constants of e by c_1. The length equation associated with e' is by definition that associated with e.

The following necessary condition is easily shown:

Proposition 2.2 *If an equation e admits a solution, then its length equation admits a non-negative integer solution.* ▮

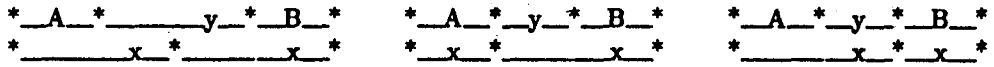
Thus solving simple equations reduces to integer programming. Next we shall see how Makanin's algorithm can solve non-simple equations.

3. Equation with scheme and position equation

In this section, we describe two basic notions appearing in Makanin's algorithm: the notion of an equation with scheme, and that of a position equation. We show how to compute a position equation from an equation with scheme.

3.1. Equation with scheme

Obviously, there are many possible ways of choosing the positions of the symbols of e_1 according to those of the symbols of e_2. For example, the following diagrams illustrate such possibilities for the equation $e = (AyB, xx)$.



Informally, a scheme applicable to an equation $e = (e_1, e_2)$ indicates how to locate the positions of the symbols of e_1 according to those of e_2 in a possible solution of e.

Formally, a **scheme** is a word $s \in = \{=,<,>\}^* =$, that is a word over the alphabet $\{=,<,>\}$ beginning and ending with the letter $=$.

A scheme s is called **applicable** to an equation $e = (e_1, e_2)$ if the following conditions are satisfied:

1) $|s|_< + |s|_= = |e_1| + 1$.
2) $|s|_> + |s|_= = |e_2| + 1$.

where $|s|_\phi$, is the number of occurrences of ϕ in s.

The left and right boundaries of a symbol t (denoted by $lb(t)$ and $rb(t)$) in a scheme s applicable to e are the integers of the interval $[1\ |s|]$, defined in the following way:

If $t = e_1[n]$ then $lb(t)$ is the length of the prefix of s whose length is equal to n over the alphabet $\{=, <\}$, and $rb(t)$ is the lehgth of the prefix of s whose length is equal to $n+1$ over $\{=, >\}$. The definition in the case $t = e_2[n]$ is obtained from the previous one by exchanging $<$ and $>$.

An **equation with scheme** is a 6-tuple (V, C, e, s, lb, rb), where e is an equation over the alphabet of variables V and the alphabet of constants C, s is a scheme applicable to e, lb and rb are left and right boundary maps.

3.2. Informal presentation of an example

An equation with scheme will now be transformed into a new object over which further processes will be applied. Before we give the formal definitions in the next sections, we introduce here the different notions and notations via an example.

Consider the equation with scheme

$$(\{A, B\}, \{x, y, z\}, AxyBz = zzx, =<><=<=, lb, rb)$$

corresponding to the following diagram:

```
*____A_____*____________x_____*___y____*____B_____*____z____*
*______________z____*_____________z____*_______________x____*
=          <        >         <        =          <         =
```

This equation with scheme will be transformed into a so called position equation. This object inherits the seven boundaries of the equation with scheme and of all occurrences of constants, but variables will be treated in a special manner.

Variables with single occurrence, like y, will disappear.

Other occurrences of variables will be renamed in order to avoid the growth of the equations appearing in the pig-pug method. The renamed occurrences of a similar variable will be associated via a symmetrical binary relation on variables (called duality relation) or a positional equivalence, depending on the number of these occurrences, as shown below:

For a two occurrence variable, like x, the two occurrences will be renamed x_1 and x_2 and associated in the duality relation. That is, $x_1 = dual(x_2)$ and $x_2 = dual(x_1)$.

For a m occurrence variable u, with m greater than two, like z, $2m-2$ renamed variables will be generated as follows :

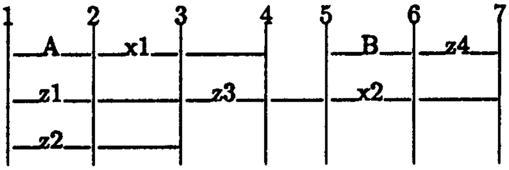
1) $m-1$ renamed variables (here z_1 and z_2) will receive the place of the first occurrence of u, (here the first occurrence of $z = e_2[1]$).

2) $m-1$ other renamed variables (here z_3 and z_4) will receive the place of the $m-1$ other occurrences of u, (here, $e_2[2]$ and $e_1[5]$).

3) each variable of the first set is in duality with one of the second set, (here, we have $dual(z_1) = z_3, dual(z_3) = z_1, dual(z_2) = z_4, dual(z_4) = z_2$).

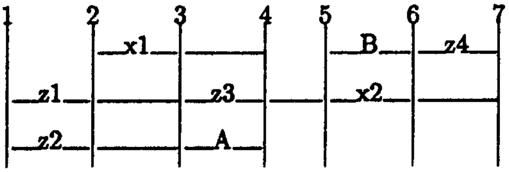
So, the duality relation gives a sort of "link" between the first $m-1$ occurrences associated with u, and the last $m-1$ occurrences. Since $z_1 = z_2$ because they have the same position, the equality of all the renamed variables $z_1 = z_2 = z_3 = z_4$ is obtained.

We then obtain an object which can be illustrated by the following diagram:

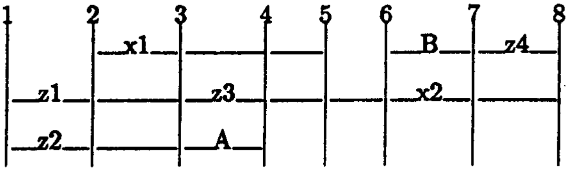


As in the pig-pug method, this position equation E will be transformed by pointing out the leftmost element (here the occurrence of A) for substituting it into the largest leftmost element (here the first largest leftmost variable z_1). The difference is that we put A at the beginning of the dual of z_1 (i.e. z_3), rather than substitute A in every variable associated with z_1. Note that, there are two ways to put A as a prefix of z_3. Either A takes all the segment between 3 and 4, or a part of this segment.

In order to avoid any loss of information during this move, a link is created between old and new positions of A in the form of a list called connection. This transformation gives rise to the two new position equations E' and E'' represented below and corresponding to the two possible positions of A relatively to the boundary 4. The link or connection (2 z_1 4) means that the prefix of z_1 ending at boundary 2 is equal to the prefix of its dual (i.e. z_3) ending at boundary 4.



(2 z1 4)



(2 z1 4)