

# Lecture Notes in Computer Science

Edited by G. Goos and J. Hartmanis

45

Mathematical Foundations  
of Computer Science 1976



Springer-Verlag  
Berlin · Heidelberg · New York

7960459

# Lecture Notes in<sup>5</sup> Computer Science

Edited by G. Goos and J. Hartmanis

45

## Mathematical Foundations of Computer Science 1976

Proceedings, 5th Symposium,  
Gdańsk, September 6-10, 1976



E7950459



Edited by A. Mazurkiewicz



Springer-Verlag  
Berlin · Heidelberg · New York 1976

TP301  
M426  
1976  
(2)

TP  
L  
Y.45

## **Editorial Board**

P. Brinch Hansen · D. Gries · C. Moler · G. Seegmüller · J. Stoer  
N. Wirth

## **Editor**

Antoni Mazurkiewicz  
Computation Centre  
Polish Academy of Sciences  
P.O.Box 22,  
00-901 Warszawa/Poland

---

AMS Subject Classifications (1970): 02C99, 02E10, 02E15, 02H10,  
18B20, 68A05, 68A10, 68A20, 68A25, 68A30, 68A45, 94A25, 94A30

---

ISBN 3-540-07854-1 Springer-Verlag Berlin · Heidelberg · New York  
ISBN 0-387-07854-1 Springer-Verlag New York · Heidelberg · Berlin

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically those of translation, re-printing, re-use of illustrations, broadcasting, reproduction by photocopying machine or similar means, and storage in data banks.

Under § 54 of the German Copyright Law where copies are made for other than private use, a fee is payable to the publisher, the amount of the fee to be determined by agreement with the publisher.

© by Springer-Verlag Berlin · Heidelberg 1976

Printed in Germany

Printing and binding: Beltz Offsetdruck, Hemsbach/Bergstr.



## FOREWORD

This volume contains papers which were contributed for presentation at the Symposium on Mathematical Foundations of Computer Science - MFCS'76, held in Gdańsk, Poland, September 6-10, 1976. This symposium is the 5th in the series of annual MFCS symposia organized in turn in Poland /every even year/ and Czechoslovakia /every odd year/ . The aim of these symposia is to promote and to develop the mathematical approach to the basic computational phenomena.

The articles in these Proceedings consist of a number of invited papers and short communications concerning mathematical results motivated by practical problems and related to:

- \*Programs and computations,
- \*Programming languages,
- \*Data bases and information retrieval systems,
- \*Analysis and complexity of algorithms,
- \*Formal languages and automata.

The scientific interest in the above topics is increasing rapidly; an example of this interest can be seen in the number of papers submitted for the Symposium. The Program Committee has been forced to reject more than half of them /sometimes valuable ones/. The main guideline for selecting papers was their originality and relevance to the subject of the Symposium.

The Symposium is being organized by the Computation Centre of the Polish Academy of Sciences in cooperation with the University of Gdańsk.

The organizers of the Symposium are grateful to all authors for their valuable contributions and to all people who helped in the organization of the Symposium. The main part of the organizational work has been done by the following members of the committee: E.Czuchajew , P.Dembiński /Vice-Chairman/, C.Góral, W.Kwasowiec, J.Leszczycowski , W.Lipski,Jr. , A.Mazurkiewicz /Symposium Chairman/, A.W.Mostowski , B.Rykaczewska, J.Winkowski /Program Chairman/. The organizers are specially indebted to J.Winkowski, who has taken the greatest part in the preparation of this volume.

The help of Springer-Verlag, which has produced these Proceedings is highly appreciated.

Antoni Mazurkiewicz

Warsaw, May 1976



CONTENTS

Invited Lecturers

K.R. Apt and J.W. de Bakker Exercises in Denotational Semantics.....	1
W. Brauer W-Automata and their Languages .....	12
J-M. Cadiou On Semantic Issues in the Relational Model of Data .....	23
E.W. Dijkstra The Effective Arrangement of Logical Systems .....	39
G. Germano and A. Maggiolo-Schettini Recursivity, Sequence Recursivity, Stack Recursivity and Semantics of Programs .....	52
J. Gruska Descriptive Complexity (of Languages)—A Short Survey .....	65
I.M. Havel On the Branching Structure of Languages .....	81
R. Kowalski (paper not received)	
U. Montanari (paper received late, see page 576)	
N.M. Nagorny Algorithms and Real Numbers .....	99
M. Nivat (paper not received)	
M. Novotný On Mappings of Machines .....	105

A. Salomaa	
Recent Results on L Systems .....	115
P.H. Starke	
Decision Problems for Multi-Tape Automata .....	124
B.A. Trakhtenbrot	
Recursive Program Schemes and Computable Functionals .....	137
E.G. Wagner, J.B. Wright, J.A. Goguen, and J.W. Thatcher	
Some Fundamentals of Order-Algebraic Semantics .....	153

### Communications

V.N. Agafonov	
On Attribute Grammars .....	169
L. Aiello, M. Aiello, G. Attardi, P. Cavallari, and G. Prini	
Formal Definition of Semantics of Generalized Control Regimes .	173
E.S. Bainbridge	
Minimal While Programs .....	180
V. Benda and K. Bendová	
On Specific Features of Recognizable Families of Languages ....	187
E. Bergmann	
On the Termination of Simple Computer Programs .....	194
G. Berthelot and G. Roucairol	
Reduction of Petri-Nets .....	202
J. Brunner and W. Wechler	
On Behaviour of R-Fuzzy Automata .....	210
A.O. Buda	
Cannonizing Reducibility Method in the Theory of Program Schemata .....	216
C. Choffrut	
Strongly Connected G-S-M Mappings Preserving Conjugation .....	224

M.P. Chytil	
Analysis of the Non-Context-Free Component of Formal Languages .....	230
M. Colombetti and E. Pagello	
Programs, Computations and Temporal Features .....	237
W. Coy	
A Note on the Automatic Generation of Inductive Assertions ...	244
A. Cremers and Th.N. Hibbard	
On the Relationship between a Procedure and its Data .....	250
M.I. Dekhtyar	
On the Relativization of Deterministic and Nondeterministic Complexity Classes .....	255
A. Dinçâ	
The Metric Properties on the Semigroups and the Languages ....	260
D. Dobkin and R.J. Lipton	
A Lower Bound of $\frac{1}{2}n^2$ on Linear Search Programs for the Knapsack Problem .....	265
W. Dörfler	
The Direct Product of Automata and Quasi-Automata .....	270
H.-D. Ehrich	
An Axiomatic Approach to Information Structures .....	277
H. Ehrig and H.-J. Kreowski	
Parallelism of Manipulations in Multidimensional Information Structures .....	284
H.J. Genrich and G. Thieler-Mevissen (paper received late, see page 588)	
S. Ginsburg and O. Mayer	
On Strict Interpretations of Grammar Forms .....	294
I. Gorun	
A Hierarchy of Context-Sensitive Languages .....	299



J. Górski	
On Consecutive Storage of Records .....	304
M. Grabowski	
A Remark on Equivalence of Mazurkiewicz's Finite - Control Algorithms over Nets .....	311
I. Hansen and J. Leszczyłowski	
Microprogram - Oriented Model of the Controlled Structure ....	317
M.D. Harrison	
Relations between Programs with Different Storage Requirements	323
F.W. von Henke	
An Algebraic Approach to Data Types, Program Verification, and Program Synthesis .....	330
M. Höpner and M. Opp	
About Three Equational Classes of Languages Built Up by Shuffle Operations .....	337
H. Huwig	
A Machine Independent Description of Complexity Classes, Definable by Nondeterministic as well as Deterministic Turing Machines with Primitive Recursive Tape or Time Bounds ..	345
K. Indermark	
Schemes with Recursion on Higher Types .....	352
J. Irlik	
Constructing Iterative Version of a System of Recursive Procedures .....	359
W. Issel	
A Method Using a Set-Theoretical Formalism to Describe the Semantics of Programming Languages .....	364
R. Jagielski	
The Cubic Search Method .....	372

R. Janicki	
Vectors of Coroutines .....	377
J. Jędrzejowicz	
One - One Degrees of Turing Machines Decision Problems .....	385
M. Karpiński (paper received late, see page 596)	
J. Kelemen	
Heuristic Representation of State-Space by Finite Automaton .	390
R. Knast	
Semigroup Characterizations of Some Language Varieties .....	395
A. Kreczmar	
On Memory Requirements of Strassen's Algorithms .....	404
W. Kwasowicz	
Determinism in Concurrent Systems .....	408
K. Lautenbach and H. Wedde	
Generating Control Mechanisms by Restrictions .....	416
J.W. Łąski	
On Diagnosis of Algorithmical Processes .....	423
B. Mikołajczak	
On Some Properties of Cyclic Automata and their Extensions ...	430
P.D. Mosses	
Compiler Generation Using Denotational Semantics .....	436
V.A. Nepomniaschy	
On Divergence Problem for Program Schemas .....	442
A. Nijholt	
On the Parsing of LL-Regular Grammars .....	446
D. Perrin	
The Characteristic Polynomial of a Finite Automaton .....	453

P. Prusinkiewicz	
Error Detection Using Regular Languages .....	458
Z. Raś	
On a Relationship between Programs of Address Machines and Mazurkiewicz Algorithms .....	465
W.P. de Roever	
Dijkstra's Predicate Transformer, Non-Determinism, Recursion, and Termination .....	472
G. Rozenberg and D. Vermeir	
Context-Free Programmed Grammars and ETOL Systems .....	482
T. Rus	
Context-Free Algebra: A Mathematical Device for Compiler Specification .....	488
P. Ružička and J. Wiedermann	
On the Lower Bound for Minimum Comparison Selection .....	495
A. Salwicki and T. Müldner	
Computational Processes Generated by Programs with Recursive Procedures and Block Structures .....	503
J. Sakarovitch	
An Algebraic Framework for the Study of the Syntactic Monoids—Application to the Group Languages .....	510
V. Yu. Sazonov	
Degrees of Parallelism in Computations .....	517
A. Schurmann	
On Adaptability of a System of Programs .....	524
A. Skowron	
A Mathematical Model of Parallel Information Processing .....	530
M.B. Smyth	
Powerdomains .....	537

J. Tiuryn	
On the Domain of Iteration in Iterative Algebraic Theories .....	544
R. Verbeek and K. Weihrauch	
The Influence of the Data Presentation on the Computational Power of Machines .....	551
J. Winkowski	
On Sequential Modelling of Non-Sequential Processes .....	559
Z. Zwinogrodzki	
Equivalence and Optimization of Recursive Program Schemata ..	569
V. Giarratana, F. Gimona, and U. Montanari	
Observability Concepts in Abstract Data Type Specification ..	576
H.J. Genrich and G. Thieler-Mevissen	
The Calculus of Facts .....	588
M. Karpiński	
Multiplicity Functions on $\omega$ -Automata .....	596

EXERCISES IN DENOTATIONAL SEMANTICS

K.R. Apt

J.W. de Bakker

Mathematisch Centrum, Amsterdam

## 1. INTRODUCTION

The present paper is a progress report about our work on semantics and proof theory of programming languages. We study a number of fundamental programming concepts occurring e.g. in the language PASCAL, viz. assignment, sequential composition, conditionals, locality, and (recursive) procedures with parameters called-by-value and called-by-variable. Our goal is the development of a formalism which satisfies two requirements

- Semantic adequacy: the definitions capture exactly the meaning attributed to these concepts in the PASCAL report.
- Mathematical adequacy: The definitions are as precise and mathematically rigorous as possible.

Of course, full semantic adequacy cannot be achieved within the scope of our paper. Thus, we were forced to omit certain aspects of the concepts concerned. What we hope to have avoided, however, is any *essential* alteration of a concept for the sake of making it more amenable to formal treatment.

Our approach follows the method of denotational semantics introduced by Scott and Strachey (e.g. in [12]). Moreover, we investigate the connections between denotational semantics and Hoare's proof theory ([6]), insofar as pertaining to the concepts mentioned above.

As main contributions of our paper we see

- The proposal of a new definition of substitution for a *subscripted* variable. This allows an extension of Hoare's axiom for assignment to the case of assignment to a subscripted variable. (This idea is described in greater detail in [2].)
- The proposal of a semantic definition and corresponding proof rule for recursive procedures with an adequate treatment of call-by-value and call-by-variable. (We believe these to be new. The proof rule is based on Scott's (or computational) induction, which is well-understood for parameterless procedures, but hardly so for procedures with parameters. In our opinion, neither the papers of Manna et al. (e.g. in [10,11]) nor those of e.g. De Bakker ([1]), Hoare ([7]), Hoare and Wirth ([8]), Igarashi, London and Luckham ([9]) give the full story on this subject.)

It will turn out that our treatment of procedures is quite complex. However, we doubt whether an approach which is *essentially* simpler is possible. Of course, we do not claim that our formalism is the last word, but the programming notions involved *are* intricate,

and we feel that essential simplification could be obtained only by changing the language.

The paper has the following outline:

*Section 2* gives the syntax of the various language constructs. Also, a careful definition of *substitution* is given which is needed for the treatment of assignment, locality and parameter passing.

*Section 3* is devoted to the definition of the denotational semantics of the five types of statements. We introduce the semantic function  $M$  which gives meaning to a statement  $S$ , in a given *environment*  $\epsilon$  (a mapping from variables to addresses) and *store*  $\sigma$  (a mapping from addresses to values), yielding a new store  $\sigma' : M(S)(\epsilon, \sigma) = \sigma'$ . For assignment, sequential composition and conditionals the definitions are fairly straightforward. It is also reasonably clear what to do about locality, but the treatment of procedures may be rather hard to follow. Some of the causes are:

- When applying the usual least fixed point approach, one has to be careful with the types (in the set-theoretical sense) of the functions involved.
- The notion of call-by-variable (the FORTRAN call-by-reference) requires a somewhat mixed action to be taken: When the actual parameter (which has to be a variable) is subscripted, the subscript is evaluated first, and then a process of substitution of the modified actual for the formal is invoked.
- The possibility of clash of variables has to be faced. (Cf. the ALGOL 60 report, sections 4.7.3.2 (Example: b int x; proc P(x); int x; b...e; ...P(x+1)...e) and 4.7.3.3 (Example: b int x; proc P; b...x...e; ...b int x; ...P...e...e.) These problems are not exactly the same as encountered in mathematical logic; in particular, they cannot simply be solved by appropriate use of the notions of free and bound occurrence and of substitution, as customary in logic.

*Section 4* introduces the proof-theoretical framework. It contains the "Exercises in denotational semantics": For each type of statement, a corresponding axiom or proof rule is given, and it is required to show its soundness. Also, a modest attempt at dealing with substitution is included. In fact, for two rules (sequential composition and conditionals) the proof is easy, for the assignment axiom we refer to [2], whereas the remaining three cases should, at the moment of writing this, be seen as conjectures since we do not yet have fully worked out proofs available. However, we are confident that the rules, perhaps after some minor modifications, will turn out to be sound.

It may be appropriate to add an indication of the restrictions we have imposed upon our investigation. There are a few minor points (such as: only one procedure declaration, i.e., not a simultaneous system; only one parameter of each of the two types, etc.). Next, things we omitted but which we do not consider essentially difficult (such as type information in declarations) and, finally, a major omission: We have no function designators in expressions, nor do we allow procedure identifiers as parameters.

There is a vast amount of literature dealing with the same issues. Many of the papers take an *operational* approach, defining semantics in terms of abstract machines.

This we wholly circumvent in the present paper, though it is in fact needed for the justification of the least fixed point approach to recursion (to be given along the lines of De Bakker [1]). Many others take their starting point in some powerful mathematical system (universal algebra, category theory), but tend to fall short of a treatment of the subtler points of the programming notions at hand. A proof-theoretic approach can be found e.g. in Hoare and Wirth [8] or Igarashi, London and Luckham [9], but we must confess not to be able to follow their treatment of procedures and parameter passing. There are also a few papers dealing with the relationship between semantics and proof theory, such as Donahue [4], Cook [3] and Gorelick [5]. Again, the approach of these papers differs from the present one. E.g., the first one omits treatment of recursion, and the other two treat locality in a way which differs from ours (cf. the block rule in our section 4). On the other hand, we recommend the papers by Cook and Gorelick for a discussion of substitution, a topic to which we pay little attention below.

## 2. SYNTAX

We present a language which is essentially a subset of PASCAL, though there are some notational variants introduced in order to facilitate the presentation. We start with the following classes of symbols:

$SV = \{x, y, z, u, \dots\}$ : the class of *simple variables*,  
 $AV = \{a, b, \dots\}$  : the class of *array variables*,  
 $B = \{n, m, \dots\}$  : the class of *integer constants*,  
 $P = \{P, Q, \dots\}$  : the class of *procedure symbols*.

For technical reasons which will become clear below (def. 2.1, def. 3.3), we assume some well-ordering of these four sets.

Using a self-explanatory variant of BNF, we now define the classes  $V$  (*variables*),  $IE$  (*integer expressions*),  $BE$  (*boolean expressions*), and  $S$  (*statements*):

$V$  (with elements  $v, w, \dots$ )  $v ::= x|a[t]$   
 $IE$  (with elements  $r, s, t, \dots$ )  $t ::= v|n|t_1+t_2|t_1*t_2| \text{if } p \text{ then } t_1 \text{ else } t_2 \text{ fi}$   
 $BE$  (with elements  $p, q, \dots$ )  $p ::= \text{true}|\text{false}|t_1=t_2|t_1>t_2|p_1 \supset p_2|p_1 \wedge p_2|\neg p$   
 $S$  (with elements  $S, S_0, \dots$ )  $S ::= v:=t|S_1; S_2| \text{if } p \text{ then } S_1 \text{ else } S_2 \text{ fi}$   
 $\quad \quad \quad \text{begin new } x; S \text{ end}|P(t, v).$

### Remarks

1. We shall use the notation  $t_1 \equiv t_2$  ( $p_1 \equiv p_2$ ,  $S_1 \equiv S_2$ ) to indicate that  $t_1$  and  $t_2$  ( $p_1$  and  $p_2$ ,  $S_1$  and  $S_2$ ) are identical sequences of symbols.
2. Whenever convenient, we shall use parentheses to enhance readability or to avoid ambiguity. Syntactic specification of this is omitted.
3. (Variables) Note that we have *simple* variables ( $x, y, z, u$ ) and *subscripted* variables ( $a[t], b[s], \dots$ ), and that an arbitrary variable  $v$  may be both simple or subscripted.

4. (Expressions) The syntax of  $IE$  and  $BE$  has been kept simple on purpose. A minor extension would be to introduce additional operations. On the other hand, the inclusion of functions designators within  $IE$  or  $BE$  presumably would constitute a major extension, requiring substantial additional analysis below.
5. (Statements) In  $S$  we have: assignment, sequential composition, conditionals, blocks, and procedure calls. The last two cases require further comment:
6. (Blocks) We restrict ourselves to declarations of simple variables without type information. This is motivated by our wish to treat declarations only insofar as needed for the analysis of parameter passing.
7. (Procedures) *Throughout the paper, we restrict ourselves to the case that we have only one procedure declaration, given in the form*

$$(2.1) \quad P \Leftarrow \text{val } x \cdot \text{var } y \cdot S_0$$

with the following conventions

- ( $\alpha$ )  $P \in P$ ,  $x, y \in SV$ ,  $S_0 \in S$ , with  $x \neq y$ .
- ( $\beta$ )  $S_0$  is the *procedure body*,  $x$  the formal value parameter,  $y$  the formal variable parameter.
- ( $\gamma$ ) In a *call*  $P(t, v)$ ,  $t$  is the actual ( $\in IE$ ) corresponding to the formal  $x$ , and  $v$  ( $\in V$ ) corresponds to  $y$ .
- ( $\delta$ ) The declaration (2.1) is assumed to be "globally" available; a call  $P(t, v)$  always refers to (2.1) as corresponding declaration.

(In PASCAL, one would write for (2.1):

procedure  $P(x:\text{integer}, \text{var } y:\text{integer}); S_0$ ).

Extension to a treatment of *systems* of declarations is reasonably straightforward (see e.g. [1]), and omitted here mainly for reasons of space; extension to any number of (value and variable) parameters is trivial.

*Substitution* plays an important role below, both in semantics and proof theory (assignment, locality, parameter mechanisms). In particular, we define

- $S[v/x]$ : substitute the (arbitrary) variable  $v$  for the simple variable  $x$  in  $S$ ;
- $s[t/v]$  and  $p[t/v]$ : substitute the integer expression  $t$  for the variable  $v$  in  $s$  or  $p$ .

The first kind of substitution is defined in the standard way using the notions of free and bound occurrence of a simple variable in a statement (An occurrence of  $x$  in  $S$  is bound whenever it is within a substatement of  $S$  of the form begin new  $x; S_1$  end. All other occurrences of  $x$  in  $S$  are free.) The second kind of substitution, which includes the case of substitution for a *subscripted* variable, was introduced in De Bakker [2]. We refer to that paper for a detailed account of this, in particular of its application in proving correctness of assignment statements.

DEFINITION 2.1. (Substitution in a statement)

- a.  $(w:=t)[v/x] \equiv (w[v/x]:=t[v/x])$
- b.  $(S_1; S_2)[v/x] \equiv (S_1[v/x]; S_2[v/x])$



- c.  $(\text{if } p \text{ then } S_1 \text{ else } S_2 \text{ fi})[v/x] \equiv \text{if } p[v/x] \text{ then } S_1[v/x] \text{ else } S_2[v/x] \text{ fi}$
- d.  $(\text{begin new } z; S \text{ end})[v/x] \equiv \text{begin new } z; S \text{ end}$ , if  $x \equiv z$   
 $\equiv \text{begin new } z; S[v/x] \text{ end}$ , if  $x \neq z$  and  $z$  does not occur free in  $v$   
 $\equiv \text{begin new } z'; S[z'/z][v/x] \text{ end}$ , if  $x \neq z$  and  $z$  occurs free in  $v$ , where  $z'$  is the first variable  $\neq x$  not occurring free in  $v$  or  $S$
- e.  $P(t, w)[v/x] \equiv P(t[v/x], w[v/x])$ .

DEFINITION 2.2. (Substitution in an expression)

- a. The definitions of  $s[t/v]$  and  $p[t/v]$  are straightforwardly reduced by formula induction to that of  $w[t/v]$ , for some  $w \in V$ .
- b. We distinguish two cases:  $v \equiv x$ , and  $v \equiv a[s]$ .
- ( $\alpha$ )  $x[t/x] \equiv t$ ,  $y[t/x] \equiv y$  ( $x \neq y$ ),  $a[s][t/x] \equiv a[s[t/x]]$
- ( $\beta$ )  $x[t/a[s]] \equiv x$ ,  $b[s'][t/a[s]] \equiv b[s'[t/a[s]]]$  ( $a \neq b$ ),  
 $a[s'][t/a[s]] \equiv \text{if } s'[t/a[s]] = s \text{ then } t \text{ else } a[s'[t/a[s]]] \text{ fi}$ .

Examples

1.  $(\text{begin new } y; x := a[y]; P(x+y+z, a[x]) \text{ end})[y/x] \equiv \text{begin new } y'; y := a[y']; P(y+y'+z, a[y]) \text{ end}$ .
2.  $x[1/a[a[1]]] \equiv x$ ,  $b[2][1/a[a[1]]] \equiv b[2]$ ,  
 $a[a[2]][1/a[a[2]]] \equiv \text{if}(\text{if } 2 = a[2] \text{ then } 1 \text{ else } a[2] \text{ fi}) = a[2]$   
 $\text{then } 1 \text{ else } a[\text{if } 2 = a[2] \text{ then } 1 \text{ else } a[2] \text{ fi}] \text{ fi}$ .  
 Observe that the last expression is semantically (section 3) (though not syntactically) equal to  $\text{if } a[2] = 2 \text{ then } a[1] \text{ else } 1 \text{ fi}$ .

### 3. DENOTATIONAL SEMANTICS

For any two sets  $K, L$ , let  $(K \rightarrow L)$  ( $(K \xrightarrow{\text{part}} L)$ ) denote the set of all functions (all *partial* functions) from  $K$  to  $L$ .

We define the meaning  $M$  of the various types of statements in our language yielding, for  $S \in S$ , as a result a partial function  $M(S)$  operating on an environment-store pair yielding a new store:  $M(S)(\epsilon, \sigma) = \sigma'$ .

As starting point we take the set  $A = \{\alpha, \beta, \dots\}$  of *addresses* and the set  $I = \{v, \mu, \dots\}$  of *integers*. Again, we assume these to be well-ordered. Let  $\Sigma = \{\sigma, \sigma', \dots\}$  be the set of *stores*, i.e.  $\Sigma = (A \rightarrow I)$ , and let  $\text{Env} = \{\epsilon, \epsilon', \dots\}$  be the set of *environments*, i.e., of certain *partial*, 1-1 functions from  $SV \cup (AV \times I)$  to  $A$ . More specifically, we require that each  $\epsilon$  is defined on a *finite* subset of  $SV$ , and on *all* elements  $AV \times I$ . Thus, for each  $x \in SV$ ,  $\epsilon(x) \in A$  may be defined, and for each  $a \in AV$  and  $v \in I$ ,  $\epsilon(a, v)$  is defined. (For a subscripted variable  $a[s]$ , if  $s$  has the current value  $v$ ,  $\epsilon(a, v)$  yields the address corresponding to  $a[s]$ . The assumption that  $\epsilon(a, v)$  is always defined stems from the fact that we study (explicit) declarations of *simple* variables only. Array variables may be considered as (implicitly) declared globally.) Next, we