Lecture Notes in Computer Science

74

Mathematical Foundations of Computer Science 1979

Proceedings. Olomouc, Czechoslovakia

Edited by J. Bečvář

Lecture Notes in Computer Science

Edited by G. Goos and J. Hartmanis

74

Mathematical Foundations of Computer Science 1979

Proceedings, 8th Symposium,
Olomouc, Czechoslovakia, September 3–7, 1979

Edited by J. Bečvář

Springer-Verlag
Berlin Heidelberg New York 1979

Editor
Jiří Bečvář
Mathematical Institute
Czechoslovak Academy of Sciences
Žitná 25
115 67 Prague 1/Czechoslovakia

AMS Subject Classifications (1970): 02E10, 02E15, 02F10, 02F15, 68A05, 68A20, 68A25, 68A30, 68A45, 68A50
CR Subject Classifications (1974):

ISBN 3-540-09526-8 Springer-Verlag Berlin Heidelberg New York
ISBN 0-387-09526-8 Springer-Verlag New York Heidelberg Berlin

Printing and binding: Beltz Offsetdruck, Hemsbach/Bergstr.
2145/3140-543210

FOREWORD

This volume contains the papers which were selected for presentation at the symposium on Mathematical Foundations of Computer Science - MFCS'79, held in Olomouc, Czechoslovakia, September 3 - 7, 1979.

The symposium is the eighth in a series of annual international meetings which take place alternately in Czechoslovakia and Poland. It has been organized by the Mathematical Institute of the Czechoslovak Academy of Sciences, Prague, the Faculty of Mathematics and Physics of Charles University, Prague, and the Faculty of Natural Sciences of Palacký University, Olomouc, in co-operation with the Federal Ministry for Technical and Investment Development, the Technical University, Prague, the Computing Research Centre, Bratislava, the Faculty of Natural Sciences of Komenský University, Bratislava, and the Faculty of Natural Sciences of Šafárik University, Košice.

The articles in these Proceedings include invited papers and short communications. The latter were selected from among 95 extended abstracts submitted in response to the call for papers. Selection was made on the basis of originality and relevance to theoretical computer science by the following Program Committee: J. Bečvář /Chairman/, J. Gruska, P. Hájek, M. Chytil, J. Král, M. Novotný, B. Rovan. A number of referees helped the Program Committee in the evaluation of the abstracts.

The papers included in these Proceedings were not formally refereed. It is anticipated that most of them will appear in a polished and completed from in scientific journals.

The organizers of the symposium are much indebted to all those who contributed to the program, in particular to the authors of the papers. Special thanks are due to the referees of the abstracts. Thanks are also due to all the above mentioned co-operating institutions for their valuable assistance and support, and to all the persons who helped in organizing the symposium. The Organizing Committee consisted of J. Bečvář, J. Gregor, J. Gruska, P. Hájek, I. Havel, Š. Hudák, M. Chytil, J. Král, F. Krutský, B. Miniberger, M. Novotný, A. Rázek, Z. Renc, B. Rovan, and M. Vlach /Chairman/. The Program Chairman acknowledges with gratitude the extensive assistance of I.M. Havel, P. Pudlák, and S. Žák in editing this volume.

The organizers of the symposium wish to express their thanks to the representatives of the Palacký University in Olomouc for their support and interest in the symposium.

Finally, the help of the Springer-Verlag in the timely publication of this volume is highly appreciated.

Prague, May 1979

Jiří Bečvář

C O N T E N T S

Invited lectures

Communications

* Manuscript received too late to be placed correctly in the alphabetic listing.

A SOUND AND COMPLETE PROOF SYSTEM FOR PARTIAL PROGRAM CORRECTNESS

J.W. de Bakker
Mathematical Centre
2e Boerhaavestraat 49, Amsterdam

1. *Introduction*

We investigate soundness and completeness of a proof system dealing with partial correctness of programs in a language with assignment, composition, conditionals, block structure, subscripted variables and (possibly recursive) procedures with the parameter mechanisms of call-by-value and call-by-address (call-by-variable in PASCAL, call-by-reference in FORTRAN). The paper is a continuation of Apt & de Bakker [3] presented at MFCS '76, and of its successor Apt & de Bakker [4]. In the meantime various problems not yet well-understood at that time have been pursued further and, we hope, solved.

Section 2 presents syntax and (denotational) semantics of our language; in section 3 we are confronted with an unpleasant consequence of our way of defining the semantics of a block b new x; S e, and propose as solution to restrict our correctness considerations to programs obeying the restriction that all such local variables x be *initialized*. Section 4 introduces the proof system; in the course of trying to prove its soundness we were somewhat shocked by the discovery that essential rules such as, for example, the familiar composition rule turned out to be invalid with respect to the natural validity definition, requiring a complicated refinement of that definition to remedy the situation. Section 5, finally, discusses the completeness of the system.

All proofs are omitted in this paper; they are scheduled to appear in a forthcoming publication.

Our paper could not have been written without Apt [1]. Though the technical results differ (e.g., [1] does not treat parameter mechanisms, nor does it impose the initialization requirement), there are many similarities, in particular concerning the validity definition and soundness proof. Also, the problem signalled at the beginning of section 3 was found by K.R. Apt. Various other approaches to the topic of our paper have appeared in the literature (Cartwright & Oppen [6], Clarke [7], Cook [8], Gorelick [9], to mention only a small selection; for a detailed survey see Apt [2]). However, out treatment of both soundness and completeness of the proposed proof system differs substantially from the techniques used elsewhere; in particular, we have not encountered any analogue of our validity definition in its refined form.

Acknowledgement. As should be clear from the above, our paper owes a lot to the work of K.R. Apt. J.I. Zucker contributed many helpful comments on a previous version.

2. *Syntax and semantics*

Convention. By writing "Let $(\alpha\epsilon)V$ be the set such that ... "we introduce a set V, with variable α ranging over V, such that ... .

2.1. *Syntax.* "≡" denotes identity between two syntactic objects. Let $(n,m,\epsilon)Icon$ be the set of integer constants, let $(x,y,z,u\epsilon)Svar$, $(a\epsilon)Avar$, $(P,Q\epsilon)Pvar$ be the (infinite, well-ordered) sets of *simple variables* (s.v.), *array variables* and *procedure variables*.

Let $(v,w\epsilon)Ivar$ be the set of *integer variables* defined by $v ::= x \mid a[s]$

... $(s,t\epsilon)Iexp$ ...	*integer expressions* ...	$s ::= n \mid v \mid s_1 + s_2 \mid$ **if** b **then** s_1 **else** s_2 **fi**
$(b\epsilon)Bexp$	*boolean expressions*	$b ::= \underline{true} \mid \underline{false} \mid s_1 = s_2 \mid \neg b \mid b_1 \supset b_2$
$(S\epsilon)Stat$	*statements*	$S ::= v := s \mid S_1 ; S_2 \mid$ **if** b **then** S_1 **else** S_2 **fi** $\mid$ b **new** x; S **e** $\mid P(t,v)$
$(D\epsilon)Decl$	*declarations*	$D ::= P_1 \Leftarrow B_1, \ldots, P_n \Leftarrow B_n$, $n \geq 0$, and, for $1 \leq i, j \leq n$, $(P_i \equiv P_j) \Rightarrow (i = j)$
$(B\epsilon)Pbod$	*procedure bodies*	$B ::= \langle$**val** x, **add** $y : S\rangle$, $x \not\equiv y$
$(R\epsilon)Prog$	*programs*	$R ::= \langle D : S\rangle$
$(p,q,r\epsilon)Assn$	*assertions*	$p ::= \underline{true} \mid \underline{false} \mid s_1 = s_2 \mid \neg p \mid p_1 \supset p_2 \mid \exists x[p]$
$(f\epsilon)Form$	*correctness formulae* (c.f.)	$f ::= p \mid \{p\}S\{q\} \mid f_1 \wedge f_2$
$(g\epsilon)Gfor$	*generalized c.f.*	$g ::= \langle D : f_1 \Rightarrow f_2\rangle$

Remarks.

1. We write $\langle D \mid S\rangle$ instead of $\langle D : S\rangle$, and similarly in B and g. If $R \equiv \langle D \mid S\rangle$, with $D \equiv \langle P_i \Leftarrow B_i\rangle_{i=1}^{n}$, and all procedure variables occurring in S or any of the B_i, $i = 1,\ldots,n$, are included in $\{P_1,\ldots,P_n\}$, we call R *closed*.
2. Our statements have local s.v. declarations, but, for simplicity's sake, no local array or procedure declarations, nor array *bounds*.
3. In $B \equiv \langle$**val** x, **add** $y \mid S\rangle$, x is the formal value parameter and y the formal address parameter, and, in $P(t,v)$, t is the actual value par. and y the actual address par. (cf. also the definition of syntactic application in 2.2).
4. $\langle D \mid f\rangle$ is short for $\langle D \mid \underline{true} \Rightarrow f\rangle$.
5. For the intended meaning of "$\Rightarrow$" in a formula g, cf. the remark on the validity of $\langle D \mid f_1 \Rightarrow f_2\rangle$ versus the soundness of $\dfrac{\langle D \mid f_1\rangle}{\langle D \mid f_2\rangle}$ below.

2.2. *Substitution and syntactic application*

Substitution is denoted by $[\cdots/\ldots]$, e.g. we use notations such as $s[t/x]$, $p[t/x]$, $S[v/x]$, $S[a'/a]$, $S[Q/P]$, etc. In case a construct contains a variable binding operator (in $\exists x[p]$ and b **new** x; S **e**, occurrences of x in p and S are bound) the usual precautions preventing clashes between free and bound variables apply. A notation such as

$s[y_i/x_i]_{i=1}^n$ implies that, for $1 \le i, j \le n$, $(x_i \equiv x_j) \Rightarrow (i=j)$. Substitution in a procedure call only affects its parameters (i.e., $P(t,v)[w/x] \equiv P(t[w/x],v[w/x])$), but not the procedure body (possibly) associated with P in the accompanying declaration D. *svar*(s), *svar*(p), *avar*(S), *pvar*(f), etc., denote the set of all *free* simple variables of s, of p, all array variables of S, all procedure variables of f, etc. Note that *svar*(<val x, add y|S>) = *svar*(S)\{x,y}. Notations such as *svar*(D,p,S,q) should be clear. We also employ the substitution s[w/v] etc., for the definition of which we refer to de Bakker [5]. Constructs which differ at most in their bound variables are called *congruent*. The congruence relation is denoted by "$\cong$".

Syntactic application is a technique of associating with a procedure body B and two actual parameters t, v, a piece of program text B(t,v) such that, for B the body of procedure P, B(t,v) embodies the meaning of P(t,v) according to the customary semantics of the parameter mechanism of call-by-value and call-by-address: let
$B \equiv$ <val x, add y|S>.

(i) $v \equiv z$. $B(t,z) \equiv$ b new u; u:=t; S[u/x][y/z]e, where u is the first s.v. not in *svar*(x,y,z,t,S)

(ii) $v \equiv a[s]$. $B(t,a[s]) \equiv$ b new u_1,u_2; u_1:=t; u_2:=s; $S[u_1/x][a[u_2]/y]e$, where u_1 (u_2) is the first (second) s.v. not in *svar*(x,y,s,t,S)

2.3. *Domains*

A $cpo(x\epsilon)C$ is a partially ordered set with least element $\perp_C$ such that each (ascending) chain $\langle x_i \rangle_{i=0}^{\infty}$ has a lub $\bigsqcup_i x_i$. Let C_1, C_2 be cpo's. If $f_1, f_2 : C_1 \to C_2$, we put $f_1 \sqsubseteq f_2$ iff $f_1(x) \sqsubseteq f_2(x)$, all $x \in C_1$. For $f: C_1 \to C_2$, we call f *monotonic* if $x_1 \sqsubseteq x_2 \Rightarrow f(x_1) \sqsubseteq f(x_2)$. A monotonic function is called *continuous* if, for each chain $\langle x_i \rangle_i$, $f(\bigsqcup_i x_i) = \bigsqcup_i f(x_i)$. Each continuous function $f: C \to C$ has a *least fixed point* $\mu f = \bigsqcup_i f^i(\perp_C)$.

Let V_0 be the set of integers, $W_0 = \{tt,ff\}$ the set of truth-values, and E_0 the (infinite, well-ordered) set of addresses. Let $(\alpha\epsilon)V = V_0 \cup \{\perp_V\}$, $(\beta\epsilon)W = W_0 \cup \{\perp_W\}$, $(e\epsilon)E = E_0 \cup \{\perp_E\}$, with $\alpha_1 \sqsubseteq \alpha_2$ iff $\alpha_1 = \perp_V$ or $\alpha_1 = \alpha_2$, and similarly for β, e. Let $(\xi\epsilon)Intv = Svar \cup (Avar \times V_0)$ be the set of *intermediate variables*, and let *intv*(s) = *svar*(s) $\cup$ {<a,α> | $a \in avar(s)$, $\alpha \in V_0$}, etc.. Let $(e\epsilon)Env = Intv \to E$ be the set of *environments* which are required to satisfy: let $dom(\epsilon) = \{\xi \in Intv \mid \epsilon(\xi) \neq \perp_E\}$, $range(\epsilon) = \{e \in E \mid \epsilon(\xi) = e$ for some $\xi \in dom(\epsilon)\}$. Then (i) ϵ is 1-1 on its domain (ii) $\epsilon(a,\alpha) \neq \perp_E$ for some $\alpha \in V_0 \Rightarrow \epsilon(a,\alpha) \neq \perp_E$ for all $\alpha \in V_0$ (iii) $Intv\backslash dom(\epsilon) \neq \emptyset$, $E_0\backslash range(\epsilon) \neq \emptyset$. Let, for $y \notin dom(\epsilon)$, $e \in E_0\backslash range(\epsilon)$, $\epsilon' \overset{df.}{=} \epsilon \cup \langle y,e \rangle$ denote the extension of ϵ such that $\epsilon'(y) = e$, and similarly for $\epsilon \cup \langle y_i,e_i \rangle_{i=1}^n$, $\epsilon \cup \langle\langle a,\alpha \rangle, e_\alpha \rangle_{\alpha \in V_0}$, etc. Let $(\sigma\epsilon)\Sigma = (E \to V) \cup \{\perp_\Sigma\}$ be the set of *states*, where, for $\sigma \in E \to V$, $\sigma(e) = \perp_V$ iff $e = \perp_E$, and $\perp_\Sigma$ ($\perp$, for short) $\overset{df.}{=} \lambda e \cdot \perp_V$. Let $\sigma\{\alpha/e\} \overset{df.}{=} \perp$, if $\sigma = \perp$, and $\lambda\bar{e}$. if $\bar{e} = e$ then α else $\sigma(\bar{e})$ fi, otherwise. Let $(\phi\epsilon)M \overset{df.}{=} Env \to (\Sigma \to \Sigma)$, $(\eta\epsilon)H \overset{df.}{=} Iexp \times Ivar \to M$, $(\gamma\epsilon)\Gamma \overset{df.}{=} Pvar \to H$, and let $\gamma\{\eta_i/P_i\}_i$ be defined similarly to $\sigma\{\alpha/e\}$. Let, finally $(\Phi\epsilon)H^n \to H$.

2.4. *Semantics*

The functions R: *Iexp* $\to (\Sigma \to V)$, L: *Ivar* $\to (\Sigma \to E)$, W: *Bexp* $\to (\Sigma \to W)$, N: *Stat* $\to (\Sigma \to \Sigma)$, M: *Prog* $\to (\Sigma \to M)$, T: *Assn* $\to (Env \to (\Sigma \to \{tt,ff\}))$, F: *Form* $\to (\Gamma \to (Env \to (\Sigma \to \{tt,ff\})))$, and G: *Gform* $\to (\Gamma \to \{tt,ff\})$ are defined by

a. If $intv(s) \notin dom(\varepsilon)$ or $\sigma = \perp$ then $R(s)(\varepsilon)(\sigma) = \perp_V$. Otherwise, $R(v)(\varepsilon)(\sigma) = \sigma(L(v)(\varepsilon)(\sigma))$, $R(m)(\varepsilon)(\sigma) = \alpha$, where α is the integer denoted by the integer constant m, $R(s_1 + s_2)(\varepsilon)(\sigma) = R(s_1)(\varepsilon)(\sigma) + R(s_2)(\varepsilon)(\sigma)$, $R(\underline{\text{if}}\ b\ \underline{\text{then}}\ s_1\ \underline{\text{else}}\ s_2\ \underline{\text{fi}})(\varepsilon)(\sigma) = \underline{\text{if}}\ W(b)(\varepsilon)(\sigma)\ \underline{\text{then}}\ R(s_1)(\varepsilon)(\sigma)\ \underline{\text{else}}\ R(s_2)(\varepsilon)(\sigma)\ \underline{\text{fi}}$.

b. If $intv(v) \notin dom(\varepsilon)$ or $\sigma = \perp$ then $L(v)(\varepsilon)(\sigma) = \perp_E$. Otherwise, $L(x)(\varepsilon)(\sigma) = \varepsilon(x)$, $L(a[s])(\varepsilon)(\sigma) = \varepsilon(a, R(s)(\varepsilon)(\sigma))$.

c. $W(b)$. Omitted.

d. If $intv(S) \notin dom(\varepsilon)$ or $\sigma = \perp$ then $N(S)(\gamma)(\varepsilon)(\sigma) = \perp$. Otherwise, $N(v:=s)(\gamma)(\varepsilon)(\sigma) = \sigma\{R(s)(\varepsilon)(\sigma)/L(v)(\varepsilon)(\sigma)\}$, $N(S_1;S_2)(\gamma)(\varepsilon)(\sigma) = N(S_2)(\gamma)(\varepsilon)(N(S_1)(\gamma)(\varepsilon)(\sigma))$, $N(\underline{\text{if}}\ \ldots\ \underline{\text{fi}}) = \ldots$, $N(\underline{b}\ \underline{\text{new}}\ x;\ \underline{S}e)(\gamma)(\varepsilon)(\sigma) = N(S[y/x])(\gamma)(\varepsilon \cup \langle y,e\rangle)(\sigma)$, where y is the first s.v. not in $dom(\varepsilon)$ and e is the first address in $E_0 \backslash range(\varepsilon)$. (*Remark.* The use of a new s.v. y ensures that we obtain the *static scope* rule for procedures.) $N(P(t,v))(\gamma)(\varepsilon)(\sigma) = \gamma(P)(t,v)(\varepsilon)(\sigma)$.

e. If $intv(R) \notin dom(\varepsilon)$ or $\sigma = \perp$ then $M(R)(\gamma)(\varepsilon)(\sigma) = \perp$. Otherwise, let $R \equiv \langle D|S\rangle$, $D \equiv \langle P_i \leftarrow B_i\rangle_{i=1}^n$. $M(R)(\gamma)(\varepsilon)(\sigma) = N(S)(\gamma\{\eta_i/P_i\}_{i=1}^n)(\varepsilon)(\sigma)$, where $\langle \eta_1,\ldots,\eta_n\rangle = \mu[\Phi_1,\ldots,\Phi_n]$, and, for $j = 1,\ldots,n$, $\Phi_j = \lambda\eta_1'.\cdots.\lambda\eta_n'.\lambda t.\lambda v$, $N(B_j(t,v))(\gamma\{\eta_i/P_i\}_i)$.

f. If $intv(p) \notin dom(\varepsilon)$ or $\sigma = \perp$ then $T(p)(\varepsilon)(\sigma) = ff$. Otherwise, $T(\underline{\text{true}})(\varepsilon)(\sigma) = tt, \ldots, T(\exists x[p])(\varepsilon)(\sigma) = \exists\alpha[T(p[y/x])(\varepsilon \cup \langle y,e\rangle)(\sigma\{\alpha/e\})]$, with $\langle y,e\rangle$ as in part d.

g. If $intv(f) \notin dom(\varepsilon)$ or $\sigma = \perp$ then $F(f)(\gamma)(\varepsilon)(\sigma) = ff$. Otherwise, $F(p)(\gamma)(\varepsilon)(\sigma) = T(p)(\varepsilon)(\sigma)$, $F(\{p\}S\{q\})(\gamma)(\varepsilon)(\sigma) = \forall\sigma'[T(p)(\varepsilon)(\sigma) \wedge \sigma' = N(S)(\gamma)(\varepsilon)(\sigma) \wedge \sigma' \neq \perp \Rightarrow T(q)(\varepsilon)(\sigma')]$, $F(f_1 \wedge f_2)(\gamma)(\varepsilon)(\sigma) = F(f_1)(\gamma)(\varepsilon)(\sigma) \wedge F(f_2)(\gamma)(\varepsilon)(\sigma)$.

h. Let $g \equiv \langle D|f_1 \Rightarrow f_2\rangle$, with $D \equiv \langle P_i \leftarrow B_i\rangle_{i=1}^n$. Let $\bar{\gamma} = \gamma\{\eta_i/P_i\}_{i=1}^n$, with η_i as in part e. $G(g)(\gamma) = [\forall\varepsilon$ such that $intv(D,f_1) \subseteq dom(\varepsilon)$, $\sigma \neq \perp\ [F(f_1)(\bar{\gamma})(\varepsilon)(\sigma)] \Rightarrow \forall\varepsilon$ such that $intv(D,f_2) \subseteq dom(\varepsilon)$, $\sigma \neq \perp\ [F(f_2)(\bar{\gamma})(\varepsilon)(\sigma)]]$.

Validity and soundness (first definition, to be modified below).

a. $\models g$ (g is valid) iff $G(g)(\gamma) = tt$ for all $\gamma \in \Gamma$

b. An *inference* $\dfrac{g_1,\ldots,g_n}{g}$ is called *sound* whenever $\models g_1,\ldots,\models g_n$ implies $\models g$.

Remark. Observe the difference between the validity of $\langle D|f_1 \Rightarrow f_2\rangle$ and soundness of $\dfrac{\langle D|f_1\rangle}{\langle D|f_2\rangle}$. Putting (i) $\overset{\text{df.}}{=\!=}$ $\forall\varepsilon$ such that $intv(D,f_i) \subseteq dom(\varepsilon)$, all $\sigma \neq \perp\ [F(f_i)(\gamma)(\varepsilon)(\sigma)]$, $i = 1,2$, we have that the former corresponds to $\forall\gamma[(1) \Rightarrow (2)]$, whereas the latter corresponds to the *weaker* fact that $\forall\gamma[(1)] \Rightarrow \forall\gamma[(2)]$.

2.5. *Lemmas.* A number of lemmas stating properties of our various constructs will be used below. First we have a lemma relating substitution to state modification.

LEMMA 2.1.

a. If $intv(s,t,v) \subseteq dom(\varepsilon)$ then $R(s[t/v])(\varepsilon)(\sigma) = R(s)(\varepsilon)(\sigma\{R(t)(\varepsilon)(\sigma)/L(v)(\varepsilon)(\sigma)\})$

b. Similarly for $b \in Bexp$, $p \in Ass n$.

END 2.1.

Next, we have a useful property of *closed* programs, asserting that such programs only affect the values of variables occurring in them:

LEMMA 2.2. If $<D|S>$ is closed, $\gamma \in \Gamma$, $\bar{\gamma}$ as usual (cf. 2.4h) then, if (i) $intv(D,S) \subseteq dom(\varepsilon)$, (ii) $\xi \in dom(\varepsilon)\backslash intv(D,S)$, (iii) $\sigma' = N(S)(\bar{\gamma})(\varepsilon)(\sigma)$, $\sigma' \neq \perp$, then, (iv) $\sigma'(\varepsilon(\xi)) = \sigma(\varepsilon(\xi))$

END 2.2.

The last lemma is rather technical, and foreshadows a property of statements to be discussed in section 3. *Notation.* For $\delta \subseteq Intv$, $(\sigma\circ\varepsilon)|\delta$ denotes the function composed of σ and ε restricted to δ.

LEMMA 2.3.

a. Let $m,n \geq 0$. If (i) $intv(s)\backslash\{x_i\}_{i=1}^{n}\backslash\{<a_j,\alpha>_{\alpha\in V_0}\}_{j=1}^{m} \subseteq \delta \subseteq dom(\varepsilon) \cap dom(\bar{\varepsilon})$, (ii) $(\sigma\circ\varepsilon)|\delta = (\bar{\sigma}\circ\bar{\varepsilon})|\delta$, (iii) For $i = 1,\ldots,n$, either $\sigma(e_i) = \bar{\sigma}(\bar{e}_i)$, or $x_i \notin svar(s)$, and, for $j = 1,\ldots,m$, either, for all $\alpha \in V_0$, $\sigma(e_{\alpha,j}) = \bar{\sigma}(\bar{e}_{\alpha,j})$, or $a_j \notin avar(s)$, then, (iv) $R(s[y_i/x_i]_i[a'_j/a_j]_j)(\varepsilon \cup <y_i,e_i>_i \cup <<<a'_j,\alpha>,e_{\alpha,j}>_\alpha>_j)(\sigma) = R(s[z_i/x_i]_i[a''_j/a_j]_j)(\bar{\varepsilon} \cup <z_i,\bar{e}_i>_i \cup <<<a''_j,\alpha>,\bar{e}_{\alpha,j}>_\alpha>_j)(\bar{\sigma})$

b. Similarly for $b \in Bexp$ and $p \in Ass n$.

END 2.3.

3. *Initialization*

The validity definition as given in 2.4 is, though rather natural, not satisfactory for our purposes. First, it implies the validity of formulae such as (*): <|{true} b new x;x:=0 e; b new x;y:=x e{y=0}>, or (**): <|{true} b new x;y:=x e; b new x;z:=x e{y=z}>. The source of this problem is that our semantics is overspecified in that, when declaring a new local s.v., we want its initial value to be some *arbitrary* integer. Now in def. 2.4d, we take for this the value stored at the first free address and, in a situation such as (*), upon entry of the second block we find, as after-effect of the first block, 0 stored at this address. ((**) can be explained similarly.) A solution to this problem is *either* to change the semantics (ensuring by some flag-mechanism that no address is ever used twice as first free address), which we do not adopt mainly because of severe technical complications, *or* to restrict our correctness considerations to programs in which all local s.v. are initialized. The second solution is the one elaborated below (also motivated by the idea that the correctness of programs containing uninitialized local s.v. is probably not very interesting anyway). A second problem with the validity definition is the following: For reasons to be explained below we have to consider in a formal correctness proof also *non-closed* programs in which case counter examples can be found to the validity of quite natural c.f. such as $<D|\{p\}S_1\{q\} \wedge \{q\}S_2\{r\} \Rightarrow \{p\}S_1;S_2\{r\}>$. The second problem is dealt with

in section 4; we now define the notion of initialization and state the main theorem concerning it.

DEFINITION 3.1. (initialized s.v.)

a. The set *init*(R) of all s.v. initialized in R is the smallest subset of *Svar* satisfying

(i) If $x \notin svar(s)$ then $x \in init(\langle D|x:=s\rangle)$.

(ii) If $x \in init(\langle D|S_1\rangle)$, or $x \notin svar(S_1)$ and $x \in init(\langle D|S_2\rangle)$ then $x \in init(\langle D|S_1;S_2\rangle)$.

(iii) If $x \notin svar(b)$, $x \in init(\langle D|S_i\rangle)$, $i = 1,2$, then $x \in init(\langle D|$ **if** b **then** S_1 **else** S_2 **fi**$\rangle)$.

(iv) If $x \neq y$, $x \in init(\langle D|S\rangle)$ then $x \in init(\langle D|$**b new** y; S **e**$\rangle)$.

(v) If $D \equiv \langle P_i \leftarrow B_i\rangle_{i=1}^n$ then, for $i = 1,\ldots,n$, if $B_i \equiv \langle$**val** x_i, **add** $y_i|S_i\rangle$, $x \notin svar(t)$, $x \equiv v$, and $y_i \in init(\langle D|S_i\rangle)$, then $x \in init(\langle D|P_i(t,v)\rangle)$.

b. All local s.v. in a program $\langle D|S\rangle$, with $D \equiv \langle P_i \leftarrow B_i\rangle_{i=1}^n$, $B_i \equiv \langle$**val** x_i, **add** $y_i|S_i\rangle$, are initialized whenever for each statement **b new** x; S_0 **e** occurring as substatement of S or any of the S_i, $1 \le i \le n$, we have that $x \in init(\langle D|S_0\rangle)$.

END 3.1.

For an initialized local s.v., the value associated with it through def. 2.4d is irrelevant. This is one of the (somewhat hidden) messages of

THEOREM 3.2. Let $\langle D|S\rangle$ be a closed program in which all local s.v. are initialized. Let $n,m \ge 0$, and let γ, $\bar\gamma$ be as usual. If

(i) $intv(D) \cup (intv(S)\backslash\{x_i\}_i\backslash\{\langle a_j,\alpha\rangle_\alpha\}_j) \subseteq \delta \subseteq dom(\varepsilon) \cap dom(\bar\varepsilon)$

(ii) $(\sigma\circ\varepsilon)|\delta = (\bar\sigma\circ\bar\varepsilon)|\delta$

(iii) For $i = 1,\ldots,n$, either $\sigma(e_i) = \bar\sigma(\bar e_i)$, or $x_i \notin svar(S)$, or $x_i \in init(\langle D|S\rangle)$.
For $j = 1,\ldots,m$, either, for all $\alpha \in V_0$, $\sigma(e_{\alpha,j}) = \bar\sigma(\bar e_{\alpha,j})$, or $a_j \notin avar(S)$.

(iv) $N(S[y_i/x_i]_i[a'_j/a_j]_j)(\bar\gamma)(\varepsilon \cup \langle y_i,e_i\rangle_i \cup \langle\langle\langle a'_j,\alpha\rangle,e_{\alpha,j}\rangle_\alpha\rangle_j)(\sigma) = \sigma'$
$N(S[z_i/x_i]_i[a''_j/a_j]_j)(\bar\gamma)(\bar\varepsilon \cup \langle z_i,\bar e_i\rangle_i \cup \langle\langle\langle a''_j,\alpha\rangle,e_{\alpha,j}\rangle_\alpha\rangle_j)(\bar\sigma) = \bar\sigma'$

then

(v) $(\sigma'\circ\varepsilon)|\delta = (\bar\sigma'\circ\bar\varepsilon)|\delta$

(vi) For $i = 1,\ldots,n$, either $\sigma'(e_i) = \bar\sigma'(\bar e_i)$, or $x_i \notin svar(S)$.
For $j = 1,\ldots,m$, either, for all $\alpha \in V_0$, $\sigma'(e_{\alpha,j}) = \bar\sigma'(\bar e_{\alpha,j})$, or $a_j \notin avar(S)$.

END 3.2.

In section 4 two special cases of this theorem are of interest, mentioned in

COROLLARY 3.3. Let $\langle D|S\rangle$, $m,n;\gamma,\bar\gamma\ldots$ be as in theorem 3.2.

a. If (i) $intv(D,S) \subseteq \delta \subseteq dom(\varepsilon) \cap dom(\bar\varepsilon)$, (ii) $(\sigma\circ\varepsilon)|\delta = (\bar\sigma\circ\bar\varepsilon)|\delta$, (iii) $N(S)(\bar\gamma)(\varepsilon)(\sigma) = \sigma'$, $N(S)(\bar\gamma)(\bar\varepsilon)(\bar\sigma) = \bar\sigma'$, then (iv) $(\sigma'\circ\varepsilon)|\delta = (\bar\sigma'\circ\bar\varepsilon)|\delta$.

b. If (i) $intv(D) \cup (intv(S)\backslash\{x_i\}_i\backslash\{\langle a_j,\alpha\rangle_\alpha\}_j) \subseteq dom(\varepsilon)$, (ii) $(S[y_i/x_i]_i[a'_j/a_j]_j)(\bar\gamma)(\varepsilon \cup \langle y_i,e_i\rangle_i \cup \langle\langle\langle a'_j,\alpha\rangle,e_{\alpha,j}\rangle_\alpha\rangle_j)(\sigma) = \sigma'$, $N(S[z_i/x_i]_i[a''_j/a_j]_j)(\gamma)(\varepsilon \cup \langle z_i,e_i\rangle_i \cup \langle\langle\langle a''_j,\alpha\rangle,e_{\alpha,j}\rangle_\alpha\rangle_j)(\sigma) = \sigma''$, then (iii) $(\sigma'\circ\varepsilon)|dom(\varepsilon) = (\sigma''\circ\varepsilon)|dom(\varepsilon)$.

END 3.3.

Remark. Let us call a pair $\langle\sigma,\varepsilon\rangle$, $\langle\bar{\sigma},\bar{\varepsilon}\rangle$ *matching* with respect to δ if it satisfies condition (ii) of part a. We see that a program satisfying the indicated requirements preserves the property of matching. Cor. 3.3b tells us that substituting different fresh s.v. y, z (since $y,z \notin dom(\varepsilon)$, $y,z \notin intv(D) \cup (intv(S)\backslash\ldots)$) for some x makes no (essential) difference in the outcome, provided that they are associated with the same address.

4. *A sound proof system*

The following proof system will be considered:

A. Rules about "⇒".

1. $\langle D|f \Rightarrow \underline{true}\rangle$ (strengthening)

2. $\dfrac{\langle D|f_1 \Rightarrow f_2\rangle}{\langle D|f_1 \wedge f_3 \Rightarrow f_2\rangle}$ (weakening)

3. $\dfrac{\langle D|f_1 \Rightarrow f_2\rangle, \langle D|f_2 \Rightarrow f_3\rangle}{\langle D|f_1 \Rightarrow f_3\rangle}$ (transitivity)

4. $\dfrac{\langle D|f \Rightarrow f_1\rangle, \langle D|f \Rightarrow f_2\rangle}{\langle D|f \Rightarrow f_2 \wedge f_2\rangle}$ (collection)

5. $\langle D|f_1 \wedge \ldots \wedge f_n \Rightarrow f_i\rangle$, $n \geq 1$, $1 \leq i \leq n$ (selection)

B. Rules about programming concepts.

6. $\langle D|\{p[t/v]\}v:=t\{p\}\rangle$ (assignment)

7. $\langle D|\{p\}S_1\{q\} \wedge \{q\}S_2\{r\} \Rightarrow \{p\}S_1;S_2\{r\}\rangle$ (composition)

8. $\langle D|\{p \wedge b\}S_1\{q\} \wedge \{p \wedge \neg b\}S_2\{q\} \Rightarrow \{p\}$ *if* ... *fi* $\{q\}\rangle$ (conditional)

9. $\langle D|\{p\}S[y/x]\{q\} \Rightarrow \{p\}\underline{b}$ *new* x; S $\underline{e}\{q\}\rangle$ (s.v. declaration)
provided that $y \notin svar(D,p,S,q)$

10. Let Ω be a procedure constant such that $N(\Omega) = \lambda\gamma\cdot\lambda t\cdot\lambda v\cdot\lambda\varepsilon\cdot\lambda\sigma\cdot\perp$.

$$\frac{\langle\langle P_i \Leftarrow B_i\rangle_i | f[\Omega/Q_i]_i\rangle, \langle\langle P_i \Leftarrow B_i\rangle_i | f \Rightarrow f[B_i'/Q_i]_i\rangle}{\langle\langle P_i \Leftarrow B_i\rangle_i | f[P_i/Q_i]_i\rangle}$$ (induction)

where $Q_i \notin pvar(P_1,\ldots,P_n,B_1,\ldots,B_n)$, and $B_i' \equiv B_i[Q_j/P_j]_j$, $i = 1,\ldots,n$.

C. Auxiliary rules

11. $\langle D|(p \supset p_1) \wedge \{p_1\}S\{q_1\} \wedge (q_1 \supset q) \Rightarrow \{p\}S\{q\}\rangle$ (consequence)