stichting
mathematisch
centrum

MC

AFDELING ZUIVERE WISKUNDE (DEPARTMENT OF PURE MATHEMATICS)

ZW 98/77

MEI

T.M.V. JANSSEN & P. VAN EMDE BOAS

THE EXPRESSIVE POWER OF INTENSIONAL LOGIC IN THE SEMANTICS OF PROGRAMMING LANGUAGES

Preprint

2e boerhaavestraat 49 amsterdam

Printed at the Mathematical Centre, 49, 2e Boerhaavestraat, Amsterdam.

The Mathematical Centre, founded the 11-th of February 1946, is a non-profit institution aiming at the promotion of pure mathematics and its applications. It is sponsored by the Netherlands Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O).

AMS(MOS) subject classification scheme (1970): 68A05, 02C10, 68A30

The expressive power of intensional logic in the semantics of programming languages *)

by

T.M.V. Janssen & P. van Emde Boas **)

ABSTRACT

A Hoare-like semantics is presented for the assignment to various types, including array elements and pointers. The semantics is based upon interpretation in a model of intensional logic. Furthermore, we obtain a general relation between forward and backward predicate transformers. By means of this relation is shown that our proposed semantics yields weakest preconditions.

KEY WORDS & PHRASES: *program correctness, assignment statement, Montague grammar, intensional logic, Hoare's assignment axiom, predicate transformers, weakest precondition, strongest postcondition, ALGOL 68, subscripted variables, pointers.*

---

*) This report will be submitted for publication elsewhere.

**) Inst. Appl. Math./VPW., Univ. of Amsterdam.

1. INTRODUCTION

The purpose of the theory of semantics of programming languages is to describe in a computer independent way those aspects of the processes taking place during execution of a program which are considered as mathematically relevant. Depending on whether the computations themselves or the results of the computations are the subject of the description one obtains operational or denotational semantics. In the latter case the result is often expressed using predicate transformers, which tell how a description of the state of the computer before execution of the program is transformed into a description of the state afterwards (forward semantics) or conversely, given a "goal"-state the transformer yields a description of a state from which the goal-state is obtained by execution of the program (backward semantics). In this approach one always aims at the forward transformation yielding as much information as possible (a strongest postcondition) or the backwards transformer yielding the least restrictive description (a weakest precondition).

For several elementary programming constructs forward and backward transformers have been described. We will consider in particular those for the assignment statement: the forward one of Floyd (1967) and the backward one of Hoare (1969). These rules are given below; with $[z/x]\phi$ is denoted the expression obtained from ϕ by replacing all occurrences of x by z.

(1) $\{\phi\}\ x:=t\ \{\exists z[[z/x]\phi \wedge x = [z/x]t]\}$ (Floyd),

(2) $\{[t/x]\psi\}\ x:=t\ \{\psi\}$ (Hoare).

Gries (1976) consideres for Hoare's rule the case

(3) $\{1=a[j]\}\ a[i]:=1\ \{a[i]=a[j]\}$.

The weakest precondition in fact should be $1 = a[j] \vee i=j$.

De Bakker (1976) considers for Floyd's rule the case

(4) $\{a[1]=2 \wedge a[2]=2\}\ a[a[2]]:=1$

$\{\exists z[[z/a[a[2]]](a[1]=2 \wedge a[2]=2) \wedge a[a[1]] = [z/a[a[2]]]1]\}$.

The postcondition reduces to $a[1]=2 \wedge a[2]=2 \wedge a[a[2]]=1$ which is a contradiction. In fact afterwards $a[a[2]]\neq 1$. He also considers Hoare's rule for this assignment

(5) $\{[1/a[a[2]]](a[a[2]]=1)\}\ a[a[2]]:=1\ \{a[a[2]]=1\}$.

The precondition reduces to 1=1 and this is not restrictive enough as we conclude from (4). Clearly the same problems arise if we consider arrays of higher dimension, e.g.

(6) q[q[2][2]][2]:=1.

Another source for problems is the use of pointers. Consider for Hoare's rule the program p:=x; x:=x+1 (in this p is a pointer):

(7) {x=x+1} p:=x; {p=x+1} x:=x+1 {p=x}.

It is impossible to satisfy the weakest precondition; while we only required that finally the value of p should be x. Floyd's rule also causes problems with pointers

(8) {x=4} p:=x; {x=4 ∧ p=x} x:=3 {∃z[z=4 ∧ p=z] ∧ x=3}.

In the final state, however, the integer value related with p is 3 and not 4! The integer value related with p is modified by x:=3 although this is hardly visible from the program text!

De Bakker (1976) gives a solution to the problems illustrated in (3), (4) and (5). Gries (1976) does so for the backward transformation only. Another solution is implicitely present in Pratt (1976). No one of them treats (6), (7) or (8).

The approach in this paper is based on work of the logician R. Montague, treating the semantics of English. A motivation for doing so was the observation concerning the role of identifiers in assignment statements.

Assume that both x and y have the value 7. Then in the assignment x := y+1 we can replace y by x or by 7 without changing the result; after execution x has the value 8. If we replace however x by y the effect is modified. So sometimes an identifier may be replaced by an expression with the same value, and sometimes not. It is striking that the same phenomenon can be observed in English. Assume that Ghica and John are married. Then "John's wife has blue eyes" is true just in case "Ghica has blue eyes". So in this context we may replace "John's wife" by "Ghica". Next consider (*) "Peter believes that Ghica has blue eyes". In this sentence we cannot freely replace "Ghica" by "John's wife", since it is not clear that Peter knows that John and Ghica are married; it is possible that Peter affirms (*) and at the same time denies "John's wife has blue eyes" since he thinks John to be married to the black-eyed Selma.

Problems concerning such expressions constitute an intriguing part of language philosophy. Many linguists, philosophers and logicians worked on attempts to deal with them. The investigations culminated in the work of R. Montague. In *The proper treatment of quantification in ordinary English* (1973) he presents the syntax and semantics of a fragment of English in which such problems are treated. References to the earlier works in this direction can be found in the introductory article of Partee (1975). We will refer in the sequel to Montague's article by "PTQ".

In the sequel of this paper we shall illustrate how Montague's framework can be used to obtain satisfactory and correct forward and backward transformers for all assignment types mentioned above. The solutions involve higher order modal logic. By using the concept of intension the problems with pointers are solved, whereas the problems with array identifiers are dealt by treating them as functions. Finally using the necessity operator from modal logic we obtained a relation between forward and backward

predicate transformers enabling us to transform the two in each other. The present paper is a sequel to Janssen & Van Emde Boas (1977) *On the proper treatment of referencing, dereferencing and assignment* (fromnow on "PTR"). In that paper forward predicate transformers are considered, while in this we focus on the backward one and the relation between both.

2. MONTAGUE's APPROACH

The basic idea in Montague's treatment of semantics is the distinction extension/intension. The extension of an expression is its value in the current world; the intension is the function yielding this value in any possible world. Consider the sentence "John walks". Its extension is a truth-value; in order to decide which one, we have to investigate the current world and find out whether John is walking or not. Its intension is the boolean function which tells us for each possible world whether John is walking or not.

The same concepts can be applied in the semantics of programming languages. The extension of an integer identifier x is the integer value of x in the present situation. Its intension is the function which for any possible world (situation) yields the corresponding integer value. In the context of programs and computers, these possible worlds may of course be considered as possible states of a computer.

The idea to distinguish between two values related to an identifier is not a new one. In denotational semantics one connects with an integer identifier an L-value (playing the role of a computer address) and an R-value (being the content of such an address). So extension corresponds with R-value, and intension has some correspondence with L-value. We wish, however, to point out that the concepts intension and extension are more general since they are defined in a model without appeal to some "address-content" structure. Therefore, intension and extension are defined (in contrast with L- and R-value) for all kinds of expressions: pointers, array identifiers as well as boolean expressions.

The mathematical language used by Montague to describe the semantics of natural language is called intensional logic (IL). This is a kind of modal logic which provides the tools for working with intensions and extensions. In PTR we have presented a formal definition of syntax and model-theory for an extension of IL which we use for describing the semantics of the assignment statement. We shall not repeat this complete definition, but present instead an informal model-theoretic exposé, illustrating the features which will be used in the sequel.

For each expression δ in the programming language we will have a corresponding expression δ'; this is called the translation of δ. For each identifier used in the programs we introduce a corresponding logical constant in IL; e.g. identifier x corresponds with constant x, so $x' = x$ (notice the different type face used). These constants will be interpreted as intensions of the corresponding type. The set of possible computer states is denoted S. For each expression ϕ in IL we denote its valuation (interpretation) in state $s \in S$ by $V_s(\phi)$.

(9) PROPERTY. If x is an simple integer identifier then $V_s(x) \in \mathbb{N}^S$. This property ex-

presses that intensions of integers are functions from states to integers. Similar properties hold for pointers and arrays.

(10) PROPERTY. If p is a pointer then $V_s(p) \in (\mathbb{N}^S)^S$.

(11) PROPERTY. If a is an array identifier then $V_s(a) \in (\mathbb{N}^{\mathbb{N}})^S$.

It is a crucial aspect of a variable that its identity remains the same while its value is changed. This is expressed by the requirement

(12) PROPERTY. If χ is some identifier, then for all states s and t: $V_s(\chi') = V_t(\chi')$.

The extension corresponding with x is denoted as $\check{}x$, so

(13) DEFINITION. $V_s(\check{}\phi) = (V(\phi))(s)$.

Consequently the assertion that (the value of) x equals 7 is denoted by $\check{}(x=7)$. So

(14) EXAMPLE. $V_s(\check{}(x=7)) \in \{T,F\}$.

The intension corresponding with an IL expression ϕ is denoted as $\hat{}\phi$. So

(15) DEFINITION. $V_s(\hat{}\phi) = \lambda t[V_t(\phi)]$.

(16) EXAMPLE. $V_s(\hat{}(\check{}x=7)) \in \{T,F\}^S$.

(17) LEMMA. $V_s(\check{}\hat{}\phi) = V_s(\phi)$.

PROOF. $V_s(\check{}\hat{}\phi) = V_s(\hat{}\phi)(s) = \lambda t[V_t(\phi)](s) = V_s(\phi)$.

(18) REMARK. In general it is not true that $V_s(\hat{}\check{}\phi) = V_s(\phi)$. For a counterexample see Janssen (1976).

(19) DEFINITION. The translation of a[n] in IL is $\hat{}((\check{}(a))(n))$.

The naive approach to states as demonstrated above does not yet provide a workable framework for treating semantics of programming languages since computer states have certain properties which have to be reflected in the model. Clearly a state $s \in S$ determines the values of all identifiers but is not guaranteed that two states where all identifiers have equal values are equal. More important the crucial effect of an assignment is that it modifies the value of a single identifier, and in order to model this assignment we have to require that the resulting state always exists. These problems have been dealt with extensively in PTR.

(20) DEFINITION. $<\chi \leftarrow V_s\delta>s$ denotes the (unique) state in which all constants except χ have the same value as in state s, and in which the value of χ equals the value of the expression δ in state s.

The essential extension to IL introduced for describing assignment is the *state switcher* $\{\delta/\check{}\chi\}$ whose meaning is described semantically by

(21) DEFINITION. $V_s(\{\delta/\check{}\chi\}\phi) = V_{<\chi \leftarrow V_s(\delta)>s}(\phi)$.

The needed properties of this substitution operator are expressed by

(22) SUBSTITUTION THEOREM. *The syntactic behaviour of the semantical defined operator* $\{\delta/\check{}\chi\}$ *is described as follows:*

1. $\{\delta/\check{}\chi\}[\phi \wedge \psi] = \{\delta/\check{}\chi\}\phi \wedge \{\delta/\check{}\chi\}\psi$ *and also for the other connectives.*
2. $\{\delta/\check{}\chi\}\exists v[\phi] = \exists v[\{\delta/\check{}\chi\}\phi]$ *provided that v does not occur in* δ *(otherwise we take an alphabetical variant). Analoguously for* $\exists z$, $\forall z$, λz.
3. $\{\delta/\check{}\chi\}\phi(\psi) = \{\delta/\check{}\chi\}\phi(\{\delta/\check{}\chi\}\psi)$.
4. $\{\delta/\chi\}\hat{}\phi = \hat{}\phi$.
5. $\{\delta/\check{}\chi\}\{\gamma/\check{}\chi\}\phi = \{\{\delta/\check{}\chi\}\gamma/\check{}\chi\}\phi$.

6. $\{\delta/\check{}\chi\}c = c$ *for any constant* c, *including* $c \equiv \chi$.

7. $\{\delta/\check{}\chi\}\check{}\chi = \delta$; $\{\delta/\check{}\chi\}\check{}c = \check{}c$ *for any constant* $c \not\equiv \chi$.

Note that in other cases $\{\delta/\check{}\chi\}\check{}\phi$ *does not reduce any further.*

(23) CONSEQUENCE. The state switcher has almost the same properties as the ordinary substitution operator.

It must be mentioned that in IL the rule of λ-conversion is no longer valid in all circumstances. Instead one has the following weaker rule:

(24) THEOREM. *Let* $[\alpha/z]\phi$ *denote the expression obtained from* ϕ *by substituting* α *for each free occurrence of* z. *If* (I) *no free occurrence of a variable in* α *becomes bound by substitution of* α *for* z *in* ϕ *and* (II) *for all states* s *and* t: $\underset{s}{V}(\alpha) = \underset{t}{V}(\alpha)$ *then we have for each state* s: $\underset{s}{V}\lambda z[\phi](\alpha) = \underset{s}{V}[\alpha/z]\phi$.

As a consequence we see that λ-conversion is allowed for arguments which are intensions like the constants corresponding to program identifiers, or state predicates (i.e., elements from $\{T,F\}^S$). For arguments not satisfying (II) wrong results may be obtained.

3. ASSIGNMENT STATEMENTS

The semantics of assignment statements is dealt with by translating them into predicate transformers. State predicates being intensions of truth values, i.e., elements of $\{T,F\}^S$ are denoted in the format $\hat{}\phi$ where ϕ is a truth value expression. Then predicate transformers become functions from $\{T,F\}^S$ to $\{T,F\}^S$, and they will be denoted in the format $\lambda P\hat{}\phi$. As seen in (24) above λ-conversion for P is allowed.

The backward predicate transformer corresponding to a simple assignment is easy to describe. Since the assignment changes the value of a single identifier the resulting state is described using a state switcher. For instance if the assignment is x:=10 the corresponding state switcher is $\{10/\check{}x\}$. If in the resulting state ψ has to hold then in the starting state $\{10/\check{}x\}\psi$ has to hold. Therefore, we introduce the following

(25) DEFINITION. Let χ be an identifier and δ some expression with translations χ' and δ' respectively. Then the *backward predicate transformer* corresponding to the assignment $\chi:=\delta$ is defined by $\lambda P\hat{}[\{\delta'/\check{}\chi'\}\check{}P]$.

Note that except for the use of extension and intension this rule is a functional variant of Hoare's original rule. Rule (25), however, can be used for pointers and arrays as well.

(26) EXAMPLE. x := x+1. The corresponding backward predicate transformer reads $\lambda P\hat{}[\{(x+1)'/\check{}(x)'\}\check{}P]$. Taking for P the assertion $\hat{}(\check{}(x)>1)$ we obtain $\hat{}[\{(x+1)'/\check{}(x)'\}\check{}\hat{}(\check{}(x) > 1)] = \hat{}(\{\check{}x+1/\check{}x\}(\check{}x > 1)) = \hat{}(x+1 > 1) = \hat{}(\check{}x > 0)$.

Programs consisting of more than one assignment are dealt with by

(27) DEFINITION. Let π_1 and π_2 be (sequences of) assignment statements translated by backward predicate transformers π_1', π_2' respectively. Then the backward predicate transformer corresponding to the program $\pi_1;\pi_2$ is defined as $\lambda P[\pi_1'(\pi_2'(P))]$.

(28) EXAMPLE. p:=x; x := x+1. Taking for P the assertion $\hat{}(\check{}p=x)$ as the argument of the backward transformation corresponding with x := x+1 we obtain: $\hat{}(\{\check{}x+1/\check{}x\}[\check{}p=x]) = \hat{}(p=x)$

(since $\check{}x$ does not occur in P). Applying the backward transformation corresponding to p:=x we obtain $\hat{}(\{x/\check{}p\}\check{}\hat{}(\check{}p=x)) = \hat{}(\{x/\check{}p\}(\check{}p=x)) = \hat{}(x=x) = \hat{}\underline{true}$. We have thus obtained, in contrast with Hoare's rule, the correct weakest precondition.

It turns out that the forward predicate transformer for the assignment $\{\chi:=\delta\}$ reads $\lambda P[\hat{}\exists z[\{z/\check{}\chi'\}\check{}P \wedge \check{}\chi'=\{z/\check{}\chi'\}\delta']]$. Except for the use of intension and extension this rule is a functional variant of Floyd's rule. For details see PTR.

In the case where the left-hand side of an assignment is a subscripted variable the new state is obtained from the old one by changing the value of a single identifier, provided we take for this identifier the corresponding array identifier. Since we treat arrays as functions this means that a new function is assigned to the array. IL contains the needed λ-expressions to denote this assignment by a state switcher.

(29) DEFINITION. The predicate transformers (forward as well as backward) corresponding to the assignment $\alpha[\mu]:=\delta$ are the same as those corresponding to

$$\alpha := \lambda n \ \underline{if}\ n = \mu \ \underline{then}\ \delta \ \underline{else}\ \alpha[n] \ \underline{fi}.$$

(30) REMARKS. The idea of considering arrays as functions appears not to be new. Gries (1976b) dues this remark to Hoare. They do not mention, however, the above reduction to simple variable assignments. Repeated application of the above definition deals with assignments to higher dimensional array elements.

(31) EXAMPLE. The predicate transformers corresponding to a[i]:=1 and a := λn[if n=i then 1 else a[n] fi] are equal. Consequently transforming the assertion $\hat{}(\check{}a(i)=\check{}a(j))$ backwards we obtain $\hat{}(\lambda n[\underline{if}\ n=i\ \underline{then}\ 1\ \underline{else}\ \check{}a(n)\ \underline{fi}])(i) = \hat{}(\lambda n[\underline{if}\ n=i\ \underline{then}\ 1\ \underline{else}\ \check{}a(n)\underline{fi}])(j)$, which reduces to $\hat{}(1=\underline{if}\ j=i\ \underline{then}\ 1\ \underline{else}\ \check{}a(j)\ \underline{fi}) = \hat{}(j=i \vee \check{}a(j)=1)$. Thus we have treated Gries' problem correctly.

(32) EXAMPLE. The predicate transformers corresponding to a[a[z]]:=1 and a := λn[if n = a[z] then 1 else a[n] fi] are equal. If we transform the assertion $\hat{}(\check{}a(\check{}a(2))=1)$ backwards, we obtain after some calculations $\hat{}[\lambda n[\underline{if}\ n=\check{}a(2)\ \underline{then}\ 1\ \underline{else}\ \check{}a(n)\ \underline{fi}](\underline{if}\ 2=\check{}a(2)\ \underline{then}\ 1\ \underline{else}\ \check{}a(2)\ \underline{fi}) = 1]$, which reduces to $\hat{}((2=\check{}a(2) \wedge \check{}a(1)=1) \vee (2\neq\check{}a(2)))$. Thus we have treated de Bakker's problem correctly.

Examples concerning the forward predicate transformer can be found in PTR.

4. STRONGEST AND WEAKEST

Predicate transformers should be well behaved. We desire that the forward transformer yields the strongest possible postcondition, and that the backward one yields the weakest possible precondition. In order to define these requirements formally, we need a second kind of semantics: an operational one.

(33) DEFINITION. An *operational semantics* " is a mapping which for each program π yields a function π"; this function gives for each starting state the corresponding final state (intuitively: the state obtained by executing the program).

(35) DEFINITION. $s \models \phi$ means $\underset{s}{V}(\phi) = T$. $\models \phi$ means that for all s: $s \models \phi$.

(36) DEFINITION. The state predicate $sp(\pi,\phi)$ is called the *strongest postcondition* with respect to program π, predicate ϕ and operational semantics " if the following two conditions are satisfied: (I) if $s \models \check{}\phi$ then $\pi"(s) \models \check{}sp(\pi,\phi)$, (II) if for all s holds that

$s \models \check{}\phi$ implies $\pi''(s) \models \check{}\eta$, then $\models \check{}sp(\pi,\phi) \rightarrow \check{}\eta$.

(36) DEFINITION. The state predicate $wp(\pi,\psi)$ is called the *weakest precondition* with respect to program π, predicate ϕ and operational semantics " if the following two conditions are satisfied: (I) if $s \models \check{}wp(\pi,\psi)$ then $\pi''(s) \models \check{}\psi$, (II) if for all s holds that $s \models \check{}\eta$ implies $\pi''(s) \models \check{}\psi$, then $\models \check{}\eta \rightarrow \check{}wp(\pi,\psi)$.

(37) REMARK. There are of course several syntactic formulations of state predicates satisfying the above definitions, but they all denote the same semantic function from states to truth values. Therefore, we feel free to speak about *the* strongest postcondition (weakest precondition). We adopt here the convention to denote this semantic function in IL by the expression $sp(\pi,\phi)$ (respectively $wp(\pi,\phi)$). We assume " fixed.

If a program can be characterized by its forward as well as by its backward transformer, there must be some relation between these characterizations. This relation is given in the next theorem. In them P and Q are variables over state predicates and $\Box$ is the necessity operator from modal logic.

(38) DEFINITION. $\underset{s}{V}(\Box\phi) = T$ iff $\forall t\ \underset{t}{V}(\phi) = T$.

(39) THEOREM. (α) $\models wp(\pi,\phi) = \hat{}\exists Q[\check{}Q \wedge \Box[\check{}sp(\pi,Q) \rightarrow \check{}\phi]]$,

(β) $\models sp(\pi,\phi) = \hat{}\forall Q[\Box[\check{}\phi \rightarrow \check{}wp(\pi,Q)] \rightarrow \check{}Q]$.

PROOF. We show that the right-hand sides satisfy cond. I and II, of the corresponding definitions (35) and (36).

(α)(cond.I): suppose $s \models \exists Q[\check{}Q \wedge \Box[\check{}sp(\pi,Q) \rightarrow \check{}\phi]]$. Then for some ψ $s \models \check{}\psi$ and $s \models \Box[\check{}sp(\pi,\psi) \rightarrow \check{}\phi]$. By definition (35) we conclude $\pi''(s) \models \check{}sp(\pi,\psi)$ and by definition of $\Box$ one has $\pi''(s) \models \check{}sp(\pi,\psi) \rightarrow \check{}\phi$. Therefore, $\pi''(s) \models \check{}\phi$.

(α)(cond. II): suppose for all s we have $s \models \check{}\eta$ implies $\pi''(s) \models \check{}\phi$. Then by definition (35) $\models \check{}sp(\pi,\eta) \rightarrow \check{}\phi$ and consequently $s \models \Box[\check{}sp(\pi,\eta) \rightarrow \check{}\phi]$. Taking $Q = \eta$ we conclude that $s \models \check{}\eta \rightarrow \exists Q[\check{}Q \wedge \Box[\check{}sp(\pi,Q) \rightarrow \check{}\phi]]$; since s was arbitrary, this proves (cond. II).

(β)(cond. I): suppose $s \models \check{}\phi$ and let ψ be some arbitrary predicate for which $\pi''(s) \models \Box(\check{}\phi \rightarrow wp(\pi,\eta))$. Then $s \models \check{}\phi$ implies $s \models \check{}wp(\pi,\eta)$ and by definition (36) this implies $\pi''(s) \models \check{}\eta$. This shows $\pi''(s) \models \hat{}\forall Q[\Box[\check{}\phi \rightarrow \check{}wp(\pi,Q)] \rightarrow \check{}Q]$.

(β)(cond. II): suppose for all s $s \models \check{}\phi$ implies $\pi''(s) \models \check{}\eta$. Then by definition (37) $\models \check{}\phi \rightarrow \check{}wp(\pi,\eta)$. If $t \models \forall Q[\Box[\check{}\phi \rightarrow \check{}wp(\pi,Q)] \rightarrow \check{}Q]$ we derive by taking $Q = \eta$ that $t \models \check{}\eta$. Since t was arbitrary this proves (cond. II).

(40) COROLLARY. *If for arbitrary ϕ either $wp(\pi,\phi)$ or $sp(\pi,\phi)$ can be syntactically described by predicate transformers in* IL, *then so can both.*

Clearly it is unlikely that formulas with quantification over predicates are the expressions for *wp* or *sp* one likes to handle. The importance of (39) is that it enables to prove theoretically that some expression describes $wp(\pi,\phi)$, given a method to obtain $sp(\pi,\phi)$ or conversely. For example in PTR we have shown for some particular operational semantics " that the strongest postcondition for the program $\chi := \delta$ is obtained by $sp(\chi:=\delta,\phi) = \check{}\exists z[\{z/\check{}\chi'\}\check{}\phi \wedge \check{}\chi' = \{z/\check{}\chi'\}\delta']$.

(41) THEOREM. α) $\models sp(\chi:=\delta,\phi) = \hat{}\exists z[\{z/\check{}\chi'\}\check{}\phi \wedge \check{}\chi' = \{z/\check{}\chi'\}\delta']$ *is equivalent with*

β) $\models wp(\chi:=\delta,\phi) = \hat{}\{\delta'/\check{}\chi'\}\check{}\phi$.

PROOF. $\alpha) \Rightarrow \beta)$. We have to show for arbitrary s and ψ

(I) $s \models \{\delta'/\check{}\chi'\}\check{}\psi$ iff

(II) $s \models \exists Q[\check{}Q \wedge \Box[\exists z[\{z/\check{}\chi'\}\check{}Q \wedge \check{}\chi' = \{z/\check{}\chi'\}\delta'] \rightarrow \check{}\psi]]$.

(I) ⇒ (II). Take $Q = \hat{}\{\delta'/\check{}\chi\}\check{}\psi$, then $s \models \check{}Q$. Remains to show that for t arbitrary

(*) $t \models \exists z[\{z/\check{}\chi'\}\{\delta'/\check{}\chi'\}\check{}\psi \wedge \check{}\chi' = \{z/\check{}\chi'\}\delta']$ implies $t \models \check{}\psi$. If (*) holds for $z = z_0$, then $t \models \{\{z_0/\check{}\chi'\}\delta/\check{}\chi'\}\check{}\phi \wedge \check{}\chi' = \{z_0/\check{}\chi'\}\delta'$, so $t \models \{\check{}\chi'/\check{}\chi'\}\check{}\psi$ and hence $t \models \check{}\psi$. Q.E.D.

(II) ⇒ (I). Assume (II) holds for $Q = \phi$. Then $S \models \check{}\phi$ and $s \models \Box[\exists z[\{z/\check{}\chi'\}\check{}\phi \wedge \check{}\chi' = \{z/\check{}\chi'\}\delta'] \rightarrow \check{}\psi]$. Taking $z = z_0$ such that for arbitrary t $V_t(z_0) = V_s(\check{}\chi')$ we infer $\langle\chi' \leftarrow V_s(\delta')\rangle s \models \{z_0/\check{}\chi'\}\check{}\phi$ and also $\langle\chi' \leftarrow V_s(\delta')\rangle s \models \check{}\chi = \{z_0/\check{}\chi'\}\delta$. So $\langle\chi' \leftarrow V_s(\delta')\rangle s \models \check{}\psi$, i.e., $s \models \{\delta'/\check{}\chi'\}\check{}\psi$. Q.E.D.

β) ⇒ α). We have to show that for arbitrary s and ψ

(III) $s \models \forall Q[\Box[\check{}\phi \rightarrow \{\delta'/\check{}\chi'\}\check{}Q] \rightarrow \check{}Q]$ iff

(IV) $s \models \exists z[\{z/\check{}\chi'\}\check{}\phi \wedge \check{}\chi' = \{z/\check{}\chi'\}\delta']$.

(III) ⇒ (IV). Take for Q in (III) the assertion from IV. We prove now that the antecedens of III holds; then (IV) follows immediately. So suppose $t \models \check{}\phi$. We have to prove that $t \models \exists z[\{\delta'/\check{}\chi'\}\{z/\check{}\chi'\}\check{}\phi \wedge \{\delta'/\check{}\chi'\}[\check{}\chi' = \{z/\check{}\chi'\}\delta']]$ or, equivalently $t \models \exists z[\{z/\check{}\chi'\}\check{}\phi \wedge \delta' = \{z/\check{}\chi'\}\delta']$. This becomes true if we choose z equal to $\check{}\chi'$.

(IV) ⇒ (III). Let η be some state predicate and suppose that the antecedens of (III) holds, so (V): $\models \check{}\phi \rightarrow \{\delta'/\check{}\chi'\}\check{}\eta$. We have to prove that $s \models \check{}\eta$.
From (IV) follows that for some z_0 holds $\langle\chi' \leftarrow V_s(z_0)\rangle s \models \check{}\phi$, so $\langle\chi' \leftarrow V_s(z_0)\rangle s \models \{\delta'/\check{}\chi'\}\check{}\eta$, consequently $s \models \{\{z_0/\check{}\chi'\}\delta'/\check{}\chi'\}\check{}\eta$.
Moreover, from (IV) follows $s \models \check{}\chi' = \{z_0/\check{}\chi'\}\delta'$, consequently $s \models \check{}\eta$. Q.E.D.

(42) COROLLARY. *The weakest precondition for* $\chi := \delta$ *is* $\hat{}\{\delta'/\check{}\chi'\}\check{}\phi$.

REFERENCES.

De Bakker, J.W. (1976), *Correctness proofs for assignment statements*, Report IW 55/76, Mathematical Centre, Amsterdam.

Floyd, R.W. (1967), *Assigning meanings to programs*, in: Proc. Symp. in Appl. Math. 19 (J.T. Schwartz ed.), *Math. aspects of computer sciences*, AMS. pp.19-32.

Gries, D. (1976), *The assignment statement*, unpublished manuscript, Cornell University/ Technical University Munich.

Gries, D. (1976b), *A note on multiple assignment to subscripted variables*, unpublished manuscript, Cornell University.

Hoare, C.A.R. (1969), *An axiomatic base for computer programming*, Comm. ACM, 12, pp.576-580.

Janssen, T.M.V. (1976), *A computer program for Montague grammar: theoretical aspects and proofs for the reduction rules*, in: Amsterdam papers in formal grammar 1, (J. Groenendijk & M. Stokhof eds.), *Proceedings of the Amsterdam colloquium on Montague grammar and related topics*, pp.154-176 Centrale Interfaculteit, University of Amsterdam.

Janssen, T.M.V. & P. van Emde Boas (1977), *On the proper treatment of referencing*,

dereferencing and assignment, Proc. 4th ICALP Conf. (Turku), Lecture Notes in Comp. Sci., Springer Verlag, Berlin, to appear.

Montague, R. (1973), *The proper treatment of quantification in ordinary English*, in: *Approaches to natural language* (J. Hintikka, J. Moravcsik & P. Suppes eds.); reprinted in R.H. Thomason (1974), *Formal philosophy, selected papers of Richard Montague*, Yale University press, New Haven and London, pp. 247-270.

Partee, B. (1975), *Montague grammar and transformational grammar*, Linguistic Inquiry 6, pp.203-300.

Pratt, V.R. (1976), *Semantical considerations on Floyd-Hoare logic*, in: Proc. 17th IEEE Symp. on Foundations of Computer Science, Houston, pp.109-122.