HENKIN-TYPE SEMANTICS FOR PROGRAM-SCHEMES TO TURN NEGATIVE RESULTS TO POSITIVE

Andréka,H. Németi,I.* Sain,I.

For motivations see [2],[1],[11],[10],[7],[6].

NOTATIONS (from textbooks on logic, [9],[3]).

d denotes a similarity type. I.e. d correlates arities (numbers) to function and relation symbols.

ω denotes the set of natural numbers.

$Y = \{ y_w : w \in \omega \}$ denotes the set of variables.

F_d is the set of d-type classical first order formulas with variables in Y. Cf. e.g. [5]p.22 .

τ denotes a term of type d in the usual sense of logic. See [5]p.22 or [9]p.166 Def.10.8.(ii) .

M_d denotes the class of all classical models of type d see e.g. [5] or [9]Def.11.1.

A classical model is denoted by an underlined capital like $\underset{\sim}{D}$ and its universe is denoted by the same capital without underlining.

By a "valuation of the variables" in a model $\underset{\sim}{D}$ a function $g:\omega \to D$ is understood, see [9]p.195 .

$\tau[q]_{\underset{\sim}{D}}$ denotes the value of the term τ in the model $\underset{\sim}{D}$ under the valuation q of the variables, see [5]p.27 D.13.13 or [9]Def.11.2. If τ contains no variable, then we write τ instead of $\tau[q]_{\underset{\sim}{D}}$ if $\underset{\sim}{D}$ is understood.

${}^{A}B$ denotes the set of all functions from A into B , see [9]p.7 .

$L_d = \langle F_d, M_d, \models \rangle$ is the first order language of type d , see [11],[6].

§1. SYNTAX (of program schemes)

The followings are basically the same as 4-1.1 in Manna[8].

Now we define the set P_d of d-type program-schemes.

The set Lab of "label symbols" is defined to be a fixed infinite subset of the set of constant d-type terms, i.e. d-type terms which do not contain variable symbols. Logical symbols: $\{\land, \neg, \exists, =\}$. Other symbols: $\{\leftarrow$, IF, GOTO, HALT, (,) , :$\}$. The set U_d of d-type commands is defined by: $(i: y \leftarrow \tau) \in U_d$ if $i \in$ Lab, $y \in Y$, and τ is a d-type term. (i: IF χ GOTO v) $\in U_d$ if $i, v \in$ Lab, $\chi \in F_d$ is a formula without quantifiers. (i:HALT) $\in U_d$ if $i \in$ Lab. These are the only elements of U_d.

*Mathematical Institute of the Hungarian Academy of Sciences, Budapest, Reáltanoda u. 13-15, H-1053 Hungary

By a d-type program scheme we understand a finite sequence p of commands (elements of U_d) ending with a "HALT", in which no two members have the same label, and in which the only "HALT-command" is the last one. Further, every label occurring in p is the label of some command in p. I.e. an element p of P_d is of the form
$p = \langle (i_o:u_o), \ldots, (i_n:u_n), (i_{n+1}:\text{HALT}) \rangle$, where $(i_m:u_m) \in U_d$, etc.

§2. SEMANTICS (of program schemes)

By languages with semantics we understand triples $L = \langle F, M, \models \rangle$ where F is called syntax, M is called the set of models or possible interpretations, and $\models$ is called validity, see [6],[10],[11].

A possible semantics for P_d would be the standard classical language $\langle P_d, M_d, \overset{\omega}{\models} \rangle$ where $\underset{\sim}{D} \overset{\omega}{\models} p[q]$ for some $q \in {}^{\omega}({}^{\omega}D)$ iff q is a standard trace of the program scheme p in the model $\underset{\sim}{D}$. Since Ianov, this standard semantics was used, see [8]Chap 4. The precise definition of this $\overset{\omega}{\models}$ can be found in [6],[1]. This standard (or classical) semantics might look clean and simple but it was proved in [2], [11] that it has highly undesirable features, it is anomalous and it simply cannot be a faithful mathematical model of our real programming situation.

Here we try to develop a natural semantic framework for programs and statements about programs. In trying to understand the "Programming Situation", its languages, their meanings etc., the first question is how an interpretation or model of a program scheme $p \in P_d$ should look like. The classical approach (Manna[8]) says that an interpretation or model of a program scheme is a relational structure $\underset{\sim}{D} \in M_d$ consisting of all the possible data values. The program p contains variables, say, y. The classical approach says that y denotes elements of D just as variables in classical first order logic do. Now we argue that y does not denote elements of D but rather y denotes some kinds of "locations" or "addresses" which may contain different data values (i.e. elements of D) at different points of time. Thus there is a set I of locations, a set T of time points, and a function $ext: I \times T \to D$ which tells for every location $s \in I$ and time point $b \in T$ what the content of location s is at time point b. Of course, this content ext(s,b) is a data value, i.e. it is an element of D. Time has a structure too ("later than" etc.) and data values have structure too, thus we have structures $\underset{\sim}{T}$ and $\underset{\sim}{D}$ over the sets T and D of time points and possible data values, respectively. Therefore we shall define a model or interpretation for programs $p \in P_d$ to be a four-tuple $\mathfrak{M} = \langle \underset{\sim}{T}, \underset{\sim}{D}, I, ext \rangle$ where $\underset{\sim}{T}$ and $\underset{\sim}{D}$ are the time structure and data structure resp., I is the set of locations and $ext: I \times T \rightarrow D$ is the "content of...at time..." function.

Consider e.g. the statement "y=y+1" which frequently occurs in programs. If y denotes elements of D then the interpretation of "y=y+1" is not very natural. However, if y denotes a location $s \in I$ then "y=y+1" means that the content of the location s changes during time T .

Of course, when specifying the semantics of a programming language P_d we may have ideas about how an interpretation $\mathfrak{M}$ of P_d may look like and how it may not look. These ideas may be expressed in the form of axioms about $\mathfrak{M}$. E.g. we may postulate that $\underset{\sim}{T}$ of $\mathfrak{M}$ has to satisfy the Peano Axioms of arithmetic. These axioms are easy to express since a closer investigation of $\mathfrak{M}$ defined above reveals that it is a model of classical 3-sorted logic (the sorts being T , D , and I). Thus the axioms can be formed in classical 3-sorted logic in a convenient manner to express all our ideas or postulates about the semantics of the programming language P_d under consideration. We shall call the elements of I intensions instead of locations.

DEFINITION 1 (see [2]):
Now to every similarity type d we define an associated 3-sorted similarity type td . About many-sorted logic and model theory see [9]p.483, [3]p.42. Let t denote the similarity type of Peano Arithmetic and let t be disjoint from d . The type td is defined as follows:
There are 3 sorts of td : $\bar{t}$, $\bar{d}$, $\bar{i}$ called "time", "data", and "intensions" respectively.
The operation symbols of td are the following: the operation symbols of d , those of t, and an additional operation symbol "ext" .
The sorts (or arities) of the operation symbols of td: the op.symbols of t go from sort $\bar{t}$ to sort $\bar{t}$, those of d go from sort $\bar{d}$ to sort $\bar{d}$, the op.symbol "ext" has two arguments, the first is of sort $\bar{i}$, the second is of sort $\bar{t}$, and the result or value of "ext" is of sort $\bar{d}$. Now the definition of the 3-sorted type td is completed.

$\mathbb{TL}_d = \langle TF_d, TM_d, \models \rangle$ denotes the 3-sorted language of type td , see [9],[3]. I.e. $\mathbb{TL}_d \overset{df}{=} L_{td}$, $TF_d \overset{df}{=} F_{td}$, $TM_d \overset{df}{=} M_{td}$.
Following [3]p.42 the elements $\mathfrak{M}$ of TM_d will be denoted as $\mathfrak{M} = \langle \underset{\sim}{T}, \underset{\sim}{D}, I, ext \rangle$. In more detail: An element $\mathfrak{M}$ of TM_d has

1. three universes throughout denoted by T , D , and I of sorts $\bar{t}$, $\bar{d}$, and $\bar{i}$ respectively,
2. operations "$^nT \rightarrow T$" originating from the type t , operations "$^nD \rightarrow D$" originating from d , and an operation $ext: I \times T \rightarrow D$.

Roughly speaking, $\mathfrak{M}$ consists of structures $\underset{\sim}{T}\in M_t$, $\underset{\sim}{D}\in M_d$, and an additional operation $ext: I\times T \rightarrow D$.

Conventions: The elements of TM_d are called time-models. If a time-model is denoted by $\mathfrak{M}$, then its parts are denoted as: $\mathfrak{M} = \langle \underset{\sim}{T}, \underset{\sim}{D}, I, ext \rangle$. If a program scheme is denoted by p , then its parts are denoted as $p = \langle (i_0:u_0), \dots, (i_n:u_n), (i_{n+1}:HALT) \rangle$. Throughout, $\{y_0, \dots, y_e\}$ contains all the variables occurring in the program scheme p such that y_e really occurs in p . Then we shall use y_{e+1} as the control variable of p .

Now we define the meanings of program schemes $p\in P_d$ in the 3-sorted models $\mathfrak{M}\in TM_d$.

DEFINITION 2 : Let $p\in P_d$ and $\mathfrak{M}\in TM_d$. Let $s_0, \dots, s_{e+1} \in I$ be arbitrary intensions in $\mathfrak{M}$. The sequence $\langle s_0, \dots, s_{e+1}\rangle$ of intensions is a trace of p in $\mathfrak{M}$ if the following (i) and (ii) are satisfied:

(i) $ext(s_{e+1}, 0) = i_0$.

(ii) Suppose $b\in T$ and $ext(s_{e+1}, b) = i_m$.
If $m=n+1$ then $\forall j\ (ext(s_j, b) = ext(s_j, b+1))$, else 1.) and 2.) below hold:

1.) If $u_m = "y_w \leftarrow \tau"$ then $ext(s_{e+1}, b+1) = i_{m+1}$ and for every $j \le e$
$$ext(s_j, b+1) = \begin{cases} \tau\,[\,ext(s_0, b), \dots, ext(s_e, b)] & \text{if } j=w \\ ext(s_j, b) & \text{otherwise} \end{cases} .$$

2.) If $u_m = "\text{IF } \chi \text{ GOTO } v"$ then $ext(s_j, b+1) = ext(s_j, b)$ for every $j \le e$
$$ext(s_{e+1}, b+1) = \begin{cases} v & \text{if } \underset{\sim}{D} \models \chi\,[\,ext(s_0, b), \dots, ext(s_e, b)] \\ i_{m+1} & \text{otherwise} \end{cases} .$$

Observe that a trace is nothing but a valuation of variables of sort $\bar{i}$. For a valuation $\bar{s}$ of the variables of sort $\bar{i}$ into the universe I of $\mathfrak{M}$ we define $\mathfrak{M} \models p[\bar{s}]$ iff $\bar{s}$ is a trace of p in $\mathfrak{M}$.

By now we have defined a semantics of program schemes, i.e. we have a language $\langle P_d , TM_d , \models \rangle$. For any set $Th \subseteq TF_d$ of axioms $Mod(Th) \subseteq TM_d$ denotes the class of all models of Th. Now for every set $Th \subseteq TF_d$ we have a language $PL_{Th} = \langle P_d , Mod(Th), \models \rangle$ where $\mathfrak{M} \models p[\bar{s}]$ is defined as above. We call such a language a programming language with semantics. But it is not yet a language for reasoning about programs. That comes in the next §3 .

Remark: Note that a trace $\langle s_0, \dots, s_{e+1}\rangle$ of a program $p\in P_d$ correlates to each variable y_w occurring in the program p an intension s_w . The intension $s_w \in I$ represents a function $ext(s_w, -): T \rightarrow D$. This function is the "history" of the variable y_w during an execution of the program p in the model $\mathfrak{M}$. Def.2 ensures that the sequence $\langle ext(s_0,-), \dots, ext(s_{e+1},-)\rangle$ of functions can be considered as a behaviour or "run" or "trace" of the program p in $\mathfrak{M}$. Here s_{e+1} is the intension of the "control variable".

It might look counter-intuitive to execute programs in arbitrary elements of TM_d. However, we can collect all our postulates about time into a set $Ax \subseteq TF_d$ of axioms which this way would define the class Mod(Ax) of all intended interpretations of P_d. Then we can use the language $PL_{Ax} = \langle P_d , Mod(Ax), \models \rangle$. Such a set Ax of axioms will be proposed in Def.4. If one wants to define semantics with unusual time structure e.g. parallelism, nondeterminism, interactions etc., then one can choose an Ax different from the one proposed in this paper. Such an application of the present "Explicite Time Approach" was done in recent works of Gergely and Ury.

§3. STATEMENTS ABOUT PROGRAMS

Let $\psi \in F_d$ be arbitrary. We think of (p,ψ) as stating that whenever the program p halts the formula ψ will be true.

DEFINITION 3 : For $p \in P_d$, $\psi \in F_d$, and $\mathfrak{M} \in TM_d$ we define $\mathfrak{M} \models (p,\psi)$ to hold iff for every trace $\langle s_0, \dots, s_{e+1}\rangle \in {}^{(e+2)}I$ of p in $\mathfrak{M}$ and for every $b \in T$: if $ext(s_{e+1},b)=i_{n+1}$, then $\underset{\sim}{D} \models \psi[ext(s_0,b), \dots, ext(s_e,b)]$.

By now we have defined a new language $\langle (P_d \times F_d), TM_d , \models \rangle$. From this we obtain our "Language for Reasoning about Programs" as $RL_d = \langle (P_d \times F_d) \cup TF_d , TM_d , \models \rangle$. In the language RL_d we can form axioms $Th \subseteq TF_d$ to express all our postulates about properties of time $\underset{\sim}{T}$ and processes I happening in time just as well as our postulates about the possible data structures $\underset{\sim}{D}$. In short, Th may be the definition of the semantics of a programming language. RL_d is the final language we have arrived at, we shall investigate its properties in the rest of this paper.

$Th \models (p,\psi)$ is defined as usual i.e. it means $(\forall \mathfrak{M} \in Mod(Th))\, \mathfrak{M} \models (p,\psi)$.

§4. PROPERTIES OF THE LANGUAGE RL_d

THEOREM 1 : Denote $TS_d \overset{df}{=} (P_d \times F_d) \cup TF_d$. Then the language $RL_d = \langle TS_d , TM_d , \models \rangle$ is strongly complete, i.e. for every recursively

enumerable set $Th \subseteq TS_d$, the set $\{ \varrho \in TS_d : Th \models \varrho \}$ of its consequences is recursively enumerable.
Specially: $\{ (p,\psi) \in P_d \times F_d : Th \models (p,\psi) \}$ is recursively enumerable.
Further, the language RL_d is <u>compact</u>.
Moreover, in the proof of the present theorem we gave a <u>strongly complete calculus</u> inference system for the language RL_d . [11]

As mentioned before, we may require the theory Th to contain a certain fixed set $Ax \subseteq TF_d$ of axioms expressing all our intuitive ideas about <u>time</u> and about <u>processes</u> "happening in time". (Basically the same was done by Henkin when he defined the new semantics for higher order logic.)

<u>DEFINITION 4</u> : Roughly speaking, Ax will be nothing but the Peano Axioms for the sort $\bar{t}$. Let $\varphi(x) \in TF_d$ be such that x is a variable of sort $\bar{t}$. Then we define φ^* to be the induction formula:

$$(\ [\ \varphi(0) \land \forall x (\varphi(x) \rightarrow \varphi(x+1)) \ ] \ \rightarrow \ \forall x \ \varphi(x) \ ) .$$

$\varphi(x)$ may contain <u>other free variables</u> of all sorts. They are also free in φ^*. They are the "parameters" of the induction φ^* .

Now the induction axioms are:

$IA \stackrel{df}{=} \{ \varphi^* : \varphi(x) \in TF_d , \text{ and } x \text{ is of sort } \bar{t} \}$.

Let PA denote the Peano Axioms for the sort $\bar{t}$. Now we define:

$Ax \stackrel{df}{=} PA \cup IA \cup \{ i \neq j : i,j \in Lab \text{ and } i \neq j \}$.

<u>THEOREM 2 (Uniqueness of traces)</u>:
Let $Axe = Ax \cup \{ \forall y_1 y_2 (\forall x [ext(y_1,x) = ext(y_2,x)] \rightarrow y_1 = y_2) \}$.
Let $p \in P_d$ and $\mathfrak{M} \in Mod(Axe)$ be arbitrary. Then for a fixed input $q \in {}^{(e+1)}D$, p has <u>at most one trace</u> in $\mathfrak{M}$ starting with q .

<u>Naur-Floyd-Hoare Inductive Assertions Proof Method</u>:

In [1],[6]Def.10 precise and detailed definition was given for the relation "$Th \vdash^F (p,\psi)$ " of (p,ψ)-s being <u>Floyd provable</u> from the theory $Th \subseteq F_d$. We shall use now $\vdash^F$ as defined there.

<u>THEOREM 3</u> : Let $Th \subseteq F_d$, and $(p,\psi) \in P_d \times F_d$ be arbitrary. Then

$$Th \vdash^F (p,\psi) \quad \underline{\text{implies}} \quad (Th \cup Ax) \models (p,\psi) .$$

Let d contain a disjoint copy of t . Let $PA_d \subseteq F_d$ be the set of Peano Axioms for the type d .

<u>THEOREM 4 :</u> Let $Th \subseteq F_d$, and (p,ψ) be such that $Th \supseteq PA_d$. Then

$Th \vdash^{F} (p,\psi)$ <u>is equivalent to</u> $(Th \cup Ax) \models (p,\psi)$.

<u>PROBLEM</u>: Find a nice sufficient condition instead of "$Th \supseteq PA_d$" for the above theorem to be true. It is clear that $Th \supseteq PA_d$ is not necessary, but if we simply omit it, then the theorem becomes false.

Thm.3 says that the language $RL_{Ax} = \langle \dots , Mod(Ax) , \models \rangle$ is reasonable enough, it contains <u>no</u> "impossible models" . I.e. the models of Ax <u>do not</u> contradict the Floyd proof rules for programs. Thm.4 says that Ax is a characterization of the "information contained implicitely" in the Floyd inference system.
Nonstandard models were used similarly in Cartwright-McCarthy[4].

<u>REFERENCES</u>

1. H.Andréka, I.Németi: A characterization of Floyd provable programs. Proc.Coll.Logic in Programming, Salgótarján 1978. Colloq.Math.Soc.J. Bolyai, North Holland. To appear. Abstracted in Bull.Section of Logic, Vol 7, No 3, Wroclaw 1978. p.115-121.
2. H.Andréka, I.Németi, I.Sain: Completeness problems in verification of programs and program schemes. MFCS'79 Olomuc. Springer 1979.
3. J.Barwise (ed): Handbook of Mathematical Logic. Studies in Logic and the found. of math. Vol 90, North Holland, Amsterdam 1977.
4. R.Cartwright, J.McCarthy: Recursive programs as functions in a first order theory. Preprint Stanford Univ. 1979.
5. C.C.Chang, H.J.Keisler: Model Theory, North Holland, 1973.
6. T.Gergely, M.Szőts: On the incompleteness of proving partial correctness. Acta Cybernetica, Tom 4, Fasc 1, Szeged 1979. p.45-57.
7. T.Gergely, L.Ury: Mathematical theory of programming. Manuscript.
8. Z.Manna: Mathematical theory of computation. McGraw Hill, 1974.
9. J.D.Monk: Mathematical Logic. Springer Verlag, 1976.
10. I.Németi, I.Sain: Connections between algebraic logic and initial algebra semantics of CF languages. Proc.Coll.Logic in Programming, Salgótarján 1978. Colloq.Math.Soc.J.Bolyai, North Holland. To appear.
11. I.Sain: Abstract model theory and completeness of languages. Preprint Budapest, May 1979.

Andréka H
Németi I

Fundamentals of Computation Theory FCT '79

Proceedings of the Conference on Algebraic, Arithmetic, and Categorial Methods in Computation Theory held in Berlin/Wendisch-Rietz (GDR) September 17–21, 1979

---

edited by Prof. Dr. Lothar Budach
Humboldt-Universität Berlin

Akademie-Verlag · Berlin 1979