Acta Informatica 19, 97–113 (1983)

Acta Informatica

Standard Model Semantics for DSL A Data Type Specification Language

J.A. Bergstra and J. Terlouw

Department of Computer Science, University of Leiden, Wassenaarseweg 80, NL-2333 RA Leiden
Department of Mathematics, University of Utrecht, Budapestlaan 6, NL-3584 CD Utrecht

Summary. We discuss a data type specification language DSL(Σ) which is obtained from the first order language L(Σ) for a given signature Σ by augmenting it with schemes.

A specification is a pair (Σ,IF) with IF a finite set of axioms in DSL(Σ). As semantics of such specifications we propose: standard model semantics, SMS.

We investigate DSL/SMS as a specification mechanism for data types and compare it with both first order and algebraic specifications. A concise theory of parametrized data types is developed in this context.

Introduction

There is little or no doubt that both logic and algebra are of fundamental importance for the area of data type specification.

Semantically a data type is often viewed at as an isomorphism class of an algebra (or of a collection of algebras). For optimizing algebraic manipulations relations can be perceived as 0−1-valued functions, but usually this is not essential. We conform to this in the sense that classes of many-sorted algebras (possibly involving relations) will mathematically represent data types.

Syntactically a language is required in which to express, or specify, properties of a data type. The choice of such a language is non-trivial especially because it is not clear in advance what kind of properties are to be expressed, and for what purpose a formal representation of these properties is sought. Analysing these purposes first we see three aspects:

(1) specification of a data type: this involves giving a mathematically clear description of the data type preferably using a fixed format with a well-understood semantics.

(2) using properties of a data type in order to verify that programs running on it meet their specifications.

(3) expressing aspects of efficiency and complexity of the data type.

Of these aspects (1) and (2) will concern us here. The present state of the art in specification and correctness proving strongly suggests that a formal language must be chosen for describing data types. Recalling our point of view that this choice is non-trivial, we must at least discuss both following questions.

(A) what possible choices are there?

(B) why to make a single choice at all?

The easier question is B. Fixing one formal system, at least for some period of time, is essential for developing a general theoretical framework which is so obvious and clear that it can be tought to students. Evidently this answer implies that the language to be chosen should be simple, both formally and conceptually. This rules out category theory in our view.

Then concerning question A we list some possibilities all depending on a given signature Σ (in which finitely many sorts, functions, relations and constants are named).

(1) equational logic with conditions.

This found wide-spread use after ADJ [8] and Guttag [9].

(2) first order logic $\mathscr{L}(\Sigma)$.

This language is mostly used in connection with Hoare's Logic for partial correctness.

(3) various second order formalisms:

(i) algorithmic/dynamic logic.

(ii) μ-calculus.

(iii) first order logic with schemes.

Our choice is (3) (iii): DSL(Σ) to be explained in detail later. Although, mathematically speaking there is no novelty in our proposed formalism, the point of this paper is to suggest it as an optimal formalism for data type specifications.

DSL together with its standard model semantics SMS meets the following conditions:

(1) It provides a data type specification mechanism with a well-understood semantics that is sufficiently strong for specifying single algebras as well as classes of algebras.

Equational logic with initial or final algebra semantics is excellent for single algebras, but not for classes. First order logic provides no means to describe a single infinite algebra up to isomorphism.

(2) DSL(Σ) depends on Σ but not on any programming language that constitutes an environment for the data type. This is important because the theory should have a modular structure as well as anything else.

Of course equational and first order logic share this advantage, but algorithmic logic and μ-calculus do not.

(3) DSL(Σ) presents properties of a data type, in such a way, that given a specification $(\Sigma, \mathbb{F})$, convincingly various logics can be defined:

PCL(Σ,IF): partial correctness logic
TCL(Σ,IF): total correctness logic
PEL(Σ,IF): program equivalence logic

(of course such logics depend on a programming language).

Clearly this advantage is shared by algorithmic logic and μ-calculus. For equational logic however partial correctness presents an unpleasant obstacle whereas a total correctness logic for first order specifications is even a useless idea because only trivial asserted programs are totally correct on the set of all models of a first order specification (Σ, T).

As far as PCL(Σ,IF), TCL(Σ,IF) and PEL(Σ,IF) are concerned we will not enter into details in this paper. We note that working with **while** programs over PCL(Σ,IF) is a straightforward generalization of Hoare's logic. TCL(Σ,IF) is treated in Bergstra and Klop [1] and PEL(Σ,IF) is a generalization of the systems presented in Bergstra and Klop [2] (see also Bergstra and Terlouw [3]).

We finish this introduction by giving a short survey of the structure of the paper.

1. DSL/SMS, Many Sorted Predicate Logic with Schemes and its Standard Model Semantics

1.1. Syntax

For completeness sake we will provide a full description of: signatures Σ, Ass(Σ), the first order language over (Σ) also called assertion language and Sch(Σ) the schemes over Σ. We put DSL(Σ)=Sch(Σ).

A signature Σ consists of a listing of four sets:

(*sorts*(Σ), *functions*(Σ), *relations*(Σ), *constants*(Σ)).

These sets in turn have the following structure:

sorts(Σ) is a finite set of sort names: we will use $s_1, \ldots, s_k, s$ as meta-variables ranging over *sorts*(Σ).

functions(Σ) is a finite set of function names with arity indication, with typical form:

$$f\colon s_1 \times \ldots \times s_k \to s$$

for a function name f of arity $(s_1, \ldots, s_k, s)$.

relations(Σ) is a finite set of relation names with arity; typical format:

$$r \subseteq s_1 \times \ldots \times s_k.$$

constants(Σ) is a finite set of constant names with sort:

$$c \in s$$

With $constants_s(\Sigma)$ we denote Σ's constants for sort s.

It is common to call a signature Σ algebraic if $relations(\Sigma) = \emptyset$.

Given two signatures Σ_1 and Σ_2 one obtains $\Sigma_1 \cap \Sigma_2$, $\Sigma_1 \cup \Sigma_2$ by taking component-wise unions: $\Sigma_1 \subseteq \Sigma_2$ is meant component-wise as well.

Ass(Σ), usually named L(Σ), the first order language over Σ, (containing formulae in the terminology of logic and assertions in the terminology of program correctness) deserves a detailed definition on this spot.

One starts with introducing countably many variables $x_i^s (i \in \omega)$ for each sort s in *sorts*(Σ). Then using a simultaneous induction for each sort s the collection $\mathrm{Ter}_s(\Sigma)$ of Σ-terms for sort s is defined.

(i) $constants_s(\Sigma) \subseteq \mathrm{Ter}_s(\Sigma)$,

(ii) $x_i^s \in \mathrm{Ter}_s(\Sigma)$,

(iii) if $t_1 \in \mathrm{Ter}_{s_1}(\Sigma), \ldots, t_k \in \mathrm{Ter}_{s_k}(\Sigma)$ and $f\colon s_1 \times \ldots \times s_k \to s$ in *functions*(Σ) then $f(t_1, \ldots, t_k) \in \mathrm{Ter}_s(\Sigma)$.

A superscript s (t^s) will indicate that t is a term of sort s. Step two consists of defining the atomic assertions, there are two kinds:

(i) $t_1^s = t_2^s \in A\text{-Ass}(\Sigma)$,

(ii) if $r \subseteq s_1 \times \ldots \times s_k$ in *relations*(Σ) then $r(t_1^{s_1}, \ldots, t_k^{s_k}) \in A\text{-Ass}(\Sigma)$.

Then on basis of the atomic assertions A-Ass (Σ), the following inductive clauses generate Ass(Σ):

(i) if $\varphi \in \mathrm{Ass}(\Sigma)$ then also $\neg\varphi$, $\exists x_i^s \varphi$, $\forall x_i^s \varphi \in \mathrm{Ass}(\Sigma)$,

(ii) if $\varphi_1, \varphi_2 \in \mathrm{Ass}(\Sigma)$ then also $\varphi_1 \vee \varphi_2$, $\varphi_1 \wedge \varphi_2$, $\varphi_1 \to \varphi_2 \in \mathrm{Ass}(\Sigma)$.

These definitions are perfectly standard. Less known in computing are the schemes to which attention is turned now. Schemes should enable stronger expressive power but not at the price of excessive growth in complexity of the language. *sorts*(Σ)* is the set of all finite lists of sort names; $\sigma \in sorts(\Sigma)^*$ is in fact an arity of an appropriate formula of Ass(Σ).

Now for each $\sigma \in sorts(\Sigma)^*$ we have countably many new predicate symbols

$$\varphi_i^\sigma (i \in \omega).$$

(The φ_i^σ are called scheme variables, and will range, given an interpretation A of Σ, over all possible relations of arity σ, on the given interpretation).

Sch(Σ) the collection of schemes over Σ is generated by these inductive clauses:

(i) $\mathrm{Ass}(\Sigma) \subseteq \mathrm{Sch}(\Sigma)$,

(ii) if $\sigma = (s_1, \ldots, s_k)$ then $\varphi_i(t_1^{s_1}, \ldots, t_k^{s_k}) \in \mathrm{Sch}(\Sigma)$.

(iii) if $\Phi, \Phi_1, \Phi_2 \in \mathrm{Sch}(\Sigma)$ then so are:

$$\neg\Phi, \Phi_1 \vee \Phi_2, \Phi_1 \wedge \Phi_2, \Phi_1 \to \Phi_2, \exists x_i^s \Phi, \forall x_i^s \Phi.$$

This finishes the syntactic part of the definition of our specification language $\mathrm{DSL}(\Sigma) = \mathrm{Sch}(\Sigma)$.

A DSL *specification* then is a pair:

$$(\Sigma, \mathbb{F})$$

with Σ a signature and $\mathbb{F}$ a finite subset of $\mathrm{DSL}(\Sigma)$.

A *first order specification* is a pair

$$(\Sigma, F)$$

with F a finite subset of $\mathrm{Ass}(\Sigma)$. Of course each first order specification is a DSL-specification but not vice versa.

Example. This is the prime example of a signature, together with the prime example of a scheme:

Σ_ω: *sorts* I
functions $S: I \to I$
constants $0 \in I$

$\mathbb{F}_\omega$: $S(x) \neq 0$
$S(x) = S(y) \to x = y$
IND

with IND the following scheme:

$$\varphi(0) \wedge \forall x[\varphi(x) \to \varphi(S(x))] \to \forall x\, \varphi(x)$$

1.2. Substitution in Schemes

The intended meaning of the scheme variables is that one may substitute assertions for them. For technical reasons it is convenient to allow even substitution of schemes for the scheme variables.

Definition. Let $\Phi, \Psi \in \mathrm{Sch}(\Sigma)$. Then $\Phi[\Psi/\varphi(x_1^{s_1}, \ldots, x_n^{s_n})]$ is the result of replacing each occurrence of the form $\varphi(t_1, \ldots, t_n)$ $(t_i \in \mathrm{Ter}_{s_i}(\Sigma))$ in Φ by $\Psi[t_1/x_1^{s_1}, \ldots, t_n/x_n^{s_n}]$.

(Here 'ordinary' substitution for variables $[t/x]$ is defined just as for assertions in the usual way.)

Note that there is no restriction on the actual variables that occur in Ψ and Φ.

Example. Let $\Phi = \mathrm{IND}$ and $\Psi \equiv x+y=y+x$. Then

$$\mathrm{IND}[\Psi/\varphi(x)] \equiv \Psi[0/x] \wedge \forall x(\Psi[x/x] \to \Psi[S(x)/x]) \to \forall x\, \Psi[x/x] \equiv 0+y$$
$$= y+0 \wedge \forall x(x+y=y+x \to S(x)+y=y+S(x)) \to \forall x\, x+y=y+x.$$

(Of course this example is meant within an extension signature of Σ_ω containing a function $+$.)

Definition. Let $\Phi = \Phi(\varphi_1, \ldots, \varphi_k)$ be a scheme in $\mathrm{Sch}(\Sigma)$. The meaning of Φ w.r.t. Σ, $[\![\Phi]\!]_\Sigma$ is the collection of all substitution instances of Φ obtained by substituting assertions $p_i \in \mathrm{Ass}(\Sigma)$ for the scheme variables φ_i in Φ:

$$[\![\Phi]\!]_\Sigma = \{\Phi[\vec{p}/\vec{\varphi}] \mid \vec{p} \in \mathrm{Ass}(\Sigma)\}.$$

1.3. Derivability for Schemes

Let $\vdash$ be some conventional proof system for first order logic. It is a relation

$$(\Sigma, F) \vdash \varphi$$

between first order specifications and assertions. From a syntactic point of view $\vdash$ is attractive because of its nature as a calculus.

Aiming at generalizing $\vdash$ to the case of deriving schemes from DSL specifications one observes that axioms and rules of $\vdash$ make sense in the presence of schemes just as well.

Taking $\vdash$ as a proofsystem for deriving schemes from specifications does not reflect properly the fact that $[\![\Phi]\!]_\Sigma$ is the intended meaning of Φ within a context defined by Σ.

A suitable system is found, however, by augmenting $\vdash$ with the following substitution rule

$$\frac{\Phi_1}{\Phi_1[\Phi_2/\varphi(\vec{x})]}$$

(for $\Phi_i \in \mathrm{Sch}(\Sigma)$, φ a scheme variable).

For the resulting proofsystem we use the notation $(\Sigma, \mathbb{F}) \vdash \Phi$.

Proposition. *For each specification* $(\Sigma, \mathbb{F})$ *and assertion* $\varphi \in \mathrm{Ass}(\Sigma)$ *the following are equivalent:*

(i) $(\Sigma, \mathbb{F}) \vdash \varphi$,
(ii) $(\Sigma, [\![\mathbb{F}]\!]_\Sigma) \vdash \varphi$.

Proof. Obvious.

This proposition reduces derivability for schemes to first order derivability in the case of an assertion as conclusion. The result can be generalized: let $\Phi(\varphi_1, \ldots, \varphi_k) = \Phi \in \mathrm{Sch}(\Sigma)$:

Obtain $\Sigma_{\vec{\varphi}}$ from Σ by adding the φ_i as new relation names of suitable arity. Then the following are equivalent:

(i) $(\Sigma, \mathbb{F}) \vdash \Phi$,
(ii) $(\Sigma_{\vec{\varphi}}, \mathbb{F}) \vdash \Phi$.
(iii) $(\Sigma_{\vec{\varphi}}, [\![\mathbb{F}]\!]_{\Sigma_{\vec{\varphi}}}) \vdash \Phi$.

This also represents a reduction to first order derivability because $\Phi \in \mathrm{Ass}(\Sigma_{\varphi})$.

Remark. $(\Sigma, F) \vdash \varphi$ is complete for the usual model theoretic (Tarski) semantics $(\Sigma, F) \models \varphi$. It is possible to construct a generalization $\models'$ for $\models$ in the case of schemes such that

$$(\Sigma, \mathbb{F}) \vdash \varphi \Leftrightarrow (\Sigma, \mathbb{F}) \models' \varphi$$

becomes true. Unfortunately the semantics $\models'$ does not turn DSL(Σ) into a useful specification language. Therefore we abandon $\models'$ and the completeness of $\vdash$ in favour of an alternative semantics, standard model semantics, that is explained in the next section.

1.4. Standard Model Semantics (SMS)

In this section a semantics for DSL specifications will be outlined which turns it into a quite flexible tool for data type specification.

SMS, like other semantics, provides for a given specification $(\Sigma, \mathbb{F})$ a collection of Σ-structures that match the specification. We use the notations

$$\mathrm{Mod}_s(\Sigma, \mathbb{F}) \quad \text{and} \quad \mathrm{Alg}_s(\Sigma, \mathbb{F})$$

for the collection of Σ-structures (Σ-algebras, if Σ is algebraic) that match $(\Sigma, \mathbb{F})$ in the sense of SMS.

1.4.1. Tarski's semantics. In order to obtain full precision: a Σ-structure (model, algebra) consists of a family

$$\{A_s \mid s \in \mathit{sorts}(\Sigma)\}$$

of non-empty sets serving as domains for each sort, equipped with an interpretation for all functions relations and constants in Σ.

For $f\colon s_1 \times \ldots \times s_k \to s \in \mathit{functions}(\Sigma)$, an interpretation is a function $F\colon A_{s_1} \times \ldots \times A_{s_k} \to A_s$; for $r \subseteq s_1 \times \ldots \times s_k \in \mathit{relations}(\Sigma)$ an interpretation provides a relation $R \subseteq A_{s_1} \times \ldots \times A_{s_k}$; $c \in \mathit{constants}_s(\Sigma)$ is interpreted by an element C of A_s.

$\mathrm{Mod}(\Sigma)$ is the class of all Σ-structures ($\mathrm{Alg}(\Sigma)$ for an algebraic Σ). A valuation α is a mapping which, for $A \in \mathrm{Mod}(\Sigma)$ yields to each variable x_i^s a value $\alpha^s(i)$ in the corresponding domain. Tarski's semantics, the most commonly used semantics for $\mathrm{Ass}(\Sigma)$ indeed, inductively defines a relation

$$A, \alpha \models \varphi$$

for $A \in \mathrm{Mod}(\Sigma)$, α a valuation for A and $\varphi \in \mathrm{Ass}(\Sigma)$. On top of this, $A \models \varphi$ is defined as:

$$\forall \alpha (A, \alpha \models \varphi).$$

The essence of this last step is that assertions with free variables get a definite meaning independent of any valuation.

As a preparatory step for SMS the relation

$$A \models \Phi$$

for $A \in \mathrm{Mod}(\Sigma)$, $\Phi \in \mathrm{Sch}(\Sigma)$ is required. We define:

$$A \models \Phi. \quad \text{if and only if,}$$
$$A \models [\![\Phi]\!]_\Sigma$$

(i.e. $\forall \varphi \in [\![\Phi]\!]_\Sigma\ A \models \varphi$).

Clearly $A \models \Phi$ if and only if $\Phi = \Phi(\varphi_1, \ldots, \varphi_k)$ cannot be refuted in A by substituting for the φ_i any predicates that are first order definable on A.

1.4.2. Standard model semantics. Let $\Phi = \Phi(\varphi_1, \ldots, \varphi_k)$; the signature $\Sigma_{\bar{\varphi}}$ is obtained by adding $\varphi_1, \ldots, \varphi_k$ as new relation names to Σ for appropriate arities. A structure $A' \in \mathrm{Mod}(\Sigma_{\bar{\varphi}})$ is called an expansion of $A \in \mathrm{Mod}(\Sigma)$ if A is obtained from A' by leaving away the interpretations of $\varphi_1, \ldots, \varphi_k$.

Standard model semantics is based on a relation

$$A \models_s \Phi$$

(read: Φ is standardly true of A, or: A is a standard model of Φ) with this meaning: "for every expansion $A' \in \mathrm{Mod}(\Sigma_{\bar{\varphi}})$ of A, $A' \models \Phi$".

Clearly $A \models_s \Phi$ says that no conceivable interpretation of the φ_i on A refutes Φ.

1.4.3. Notations

(i) $A \models_s \mathbb{F}$ if for all $\Phi \in \mathbb{F}$, $A \models_s \Phi$.

(ii) $\mathrm{Mod}_s(\Sigma, \mathbb{F}) = \{A \in \mathrm{Mod}(\Sigma) \mid A \models_s \mathbb{F}\}$

(iii) $(\Sigma, \mathbb{F}) \models \Phi$: for each $A \in \mathrm{Mod}_s(\Sigma, \mathbb{F})$ $A \models_s \Phi$.

(iv) Let K and L be two classes of Σ-structures, then $K \cong L$ abbreviates: for each $A \in K$ there is a $B \in L$ with $A \cong B$ and conversely. Now if $\mathrm{Mod}_s(\Sigma, \mathbb{F}) \cong \{A\}$ we propose the more suggestive notation

$$(\Sigma, \mathbb{F}) \rightarrow A.$$

Example. $(\Sigma_\omega, \mathbb{F}_\omega) \rightarrow (\omega, S, 0)$.

1.5. *Examples of* DSL/SMS *specifications*

1.5.1. Finite sets. Take Σ_A: *sorts*: A, no functions relations or constants. $\mathrm{Mod}(\Sigma_A)$ is just the collection of all sets.

With $\mathbb{F}_{fA}$ as below, $\mathrm{Mod}_s(\Sigma_A, \mathbb{F}_{fA})$ is the collection of all finite sets. $\mathbb{F}_{fA}$ contains one scheme variable $\phi(x, y)$:

$$\mathbb{F}_{fA} \equiv \mathrm{Fun}(\phi) \wedge \mathrm{Inj}(\phi) \rightarrow \mathrm{Surj}(\phi).$$

Here: $\mathrm{Fun}(\phi) \equiv \forall x \exists! y\, \phi(x, y)$

$\mathrm{Inj}(\phi) \equiv \forall x\, y\, z [\phi(x, y) \wedge \phi(y, z) \rightarrow z = y]$

$\mathrm{Surj}(\phi) \equiv \forall y \exists x\, \phi(x, y)$.

1.5.2. Natural numbers. $N=(\omega, S, +, \cdot, 0)$, signature $\Sigma_{\omega,+}$; let $\mathbb{F}_{\omega,+}$ be

$$\mathbb{F}_{\omega} \cup \{x+0=x,\ x+S(y)=S(x+y),\ x\cdot 0=0,\ x\cdot S(y)=x\cdot y+x\}$$

Now $(\Sigma_{\omega,+}, \mathbb{F}_{\omega,+}) \rightarrow N$.

1.5.3. All bounded counters. All structures of the form $(\{0,\ldots,n\}, S_n, 0)$ with

$$S_n(i)=\begin{cases} i+1 & \text{if } i<n \\ n & \text{if } i=n. \end{cases}$$

Let Σ_{CO} be a suitable signature then $\mathrm{Mod}_s(\Sigma_{CO}, \mathbb{F}_{BCO})$ contains exactly all structures isomorphic to a bounded counter, with $\mathbb{F}_{BCO}$ as follows:

$$\begin{cases} 0 \neq S(x),\ \exists x\, F(x)=x,\ x\neq y \wedge F(x)=F(y) \rightarrow (x=F(y) \vee y=F(x)) \\ \mathrm{IND} \end{cases}$$

1.5.4. Well-ordered sets. Let Σ_{WOA} consist of one sort A and a binary relation R on it. Take $\mathbb{F}_{WOA}$:

$$\begin{cases} \forall x y z [R(x,y) \wedge R(y,z) \rightarrow R(x,z)] \\ \forall x y\, R(x,y) \vee R(y,x) \vee x=y \\ \exists x\, \varphi(x) \rightarrow \exists x(\varphi(x) \wedge \forall y[\varphi(y) \rightarrow \neg R(y,x)]). \end{cases}$$

Then $\mathrm{Mod}_s(\Sigma_{WOA}, \mathbb{F}_{WOA})$ contains all structures (A, R) with R a well-ordering of A.

1.5.5. Uncountable sets plus integers. Let $\Sigma_{\omega A}$ be a signature that extends Σ_{ω} with a new sort A. We look for a specification $(\Sigma_{\omega A}, \mathbb{F}_{\omega AL})$ such that

$$\mathrm{Mod}_s(\Sigma_{\omega A}, \mathbb{F}_{\omega AL})$$

contains all structures of the form $((\omega, S, 0), A)$ with A an uncountable set.

$\mathbb{F}_{\omega AL}=\mathbb{F}_{\omega} \cup \{\mathrm{Fun}(\phi) \rightarrow \neg \mathrm{Surj}(\phi)\}$ where $\phi(x^{\mathrm{I}}, y^{\mathrm{A}})$ is a binary scheme variable.

1.5.6. Sets on A. We aim at a specification of all structures of the form

$$(A, \mathrm{SOA}, \emptyset, E, \mathrm{INS})$$

with A a set, SOA another set with constant $\emptyset$ and operation
INS: $A \times \mathrm{SOA} \rightarrow \mathrm{SOA}$ and E a relation $\subseteq A \times \mathrm{SOA}$, such that the system is isomorphic with the one obtained by taking
SOA: finite subsets of A, $\emptyset$ the empty set, E: 'element of' and
INS: insertion.

$$\mathbb{F}_{\mathrm{SOA}}:\ \neg E(x,\emptyset),\ \forall x(E(x,X) \leftrightarrow E(x,Y)) \rightarrow X=Y$$
$$E(x, \mathrm{INS}(y,X)) \leftrightarrow x=y \vee E(x,X)$$
$$\phi(\emptyset) \wedge \forall x X(\phi(X) \rightarrow \phi(\mathrm{INS}(x,X))) \rightarrow \forall X\, \phi(X)$$

here x, y range over A; X, Y over SOA.

1.5.7. Binary trees. $(\Sigma_{BT}, \mathbb{F}_{BT})$ with

Σ_{BT}: *sorts* T
functions COMB: $T \times T \to T$
LEFT: $T \to T$
RIGHT: $T \to T$
relations $\mathrm{AT} \subseteq T$
constants $e \in T$

$\mathbb{F}_{BT}$: $\mathrm{AT}(x) \to \mathrm{LEFT}(x) = \mathrm{RIGHT}(x) = e$
$\mathrm{AT}(e)$
$\neg \mathrm{AT}(x) \leftrightarrow \exists y z (x = \mathrm{COMB}(y, z))$
$\mathrm{COMB}(x, y) = \mathrm{COMB}(v, w) \to x = v \wedge y = w$
$\mathrm{LEFT}(\mathrm{COMB}(x, y)) = x$
$\mathrm{RIGHT}(\mathrm{COMB}(x, y)) = y$
$[\forall x(\mathrm{AT}(x) \to \phi(x)) \wedge \forall x y(\phi(x) \wedge \phi(y) \to \phi(\mathrm{COMB}(x, y)))]$
$\to \forall x\, \phi(x)$.

1.6. Comparison with Initial and Final Algebra Semantics

Algebraic specifications and its two favourite semantics have been extensively studied in the literature [8-11, 13]. Given an algebraic signature Σ, an equation is a form

$$t_1^s = t_2^s \quad (s \in sorts(\Sigma), t_i \in \mathrm{Ter}_s(\Sigma))$$

and a conditional equation is a form

$$t_1^{s_1} = r_1^{s_1} \wedge \ldots \wedge t_k^{s_k} = r_k^{s_k} \to t^s = r^s \quad (s, s_i \in sorts(\Sigma), t_i, r_i, t, r \in \mathrm{Ter}(\Sigma)).$$

An algebraic specification is a specification

$$(\Sigma, F)$$

with F a finite set of conditional equations.

1.6.1. Initial and final algebra semantics. Let T_Σ denote the algebra of closed Σ-terms. Clearly T_Σ is Σ-algebra. If $\equiv$ is a congruence on T_Σ then $T_{\Sigma/\equiv}$ is also a Σ-algebra. A Σ-algebra A is minimal if for some congruence $\equiv$ or T_Σ, $A \cong T_{\Sigma/\equiv}$. We say that $\equiv$ satisfies E (is an E-congruence) if $T_{\Sigma/\equiv} \models E$.

There exists an E-congruence $\equiv_E$ on T_Σ which is the smallest one among the E-congruences. $T_{\Sigma/\equiv_E} = T(\Sigma, E)$ is the initial algebra for (Σ, E).

A congruence $\equiv$ on T_Σ is trivial if it has for each sort exactly one congruence class. Now it may or may not be the case that there exists a largest non-trivial E-congruence. If so denote it $\equiv_E^L$ and write $F(\Sigma, E) = T_{\Sigma/\equiv_E^L}$. If $F(\Sigma, E)$ exists it is called the final algebra semantics of the specification (Σ, E).

Thus, summing up: for an algebraic specification (Σ, E)

(i) there *always* exists an initial algebra semantics $T(\Sigma, E)$,
(ii) there *may* exist a final algebra semantics $F(\Sigma, E)$,
(iii) both $T(\Sigma, E)$ and $F(\Sigma, E)$ are *minimal* algebras.

1.6.2. DSL/SMS *and final algebras.* The first important observation concerning DSL/SMS in connection with algebraic specifications is the following:

Proposition. *For each algebraic signature* Σ *there is a scheme* $\mathrm{gen}(\Sigma)$ *with the property that* $\mathrm{Mod}_s(\mathrm{gen}(\Sigma))$ *is just the class of minimal* Σ*-algebras.*

Proof. Write $S = sorts(\Sigma)$; take for $s \in S$ $\phi_s(x^s)$ a unary scheme variable with variable x^s ranging over s. Then $\mathrm{gen}(\Sigma)$ is as follows:

$$[\bigwedge_{s\in S}\bigwedge_{c\in S}\phi_s(c)$$

$$\wedge \bigwedge_{f: s_1\times\ldots\times s_k\to s} (\forall x^{s_1},\ldots,x^{s_k}\,\phi_{s_1}(x^{s_1})\wedge\ldots\wedge\phi_{s_k}(x^{s_k})\to\phi_s(f(x_1^{s_1},\ldots,x_k^{s_k}))]$$

$$\Rightarrow \bigwedge_{s\in S}\forall x^s\,\phi_s(x^s).$$

Now suppose $A \models_s \mathrm{gen}(\Sigma)$. Interpret $\phi^s(x^s)$ by the predicate "x^s is the value of a closed Σ-term". It follows that A is minimal.

Consequently DSL/SMS is fully able to express that part of initial/final algebra semantics which has to do with the fact that these semantices are minimal algebras.

Considering final algebra semantics, it turns out that final algebra semantics can be conceived as a subcase of SMS when appropriate extra schemes $\mathrm{gen}(\Sigma)$ and $\max(\Sigma)$ are used.

There is a uniform construction $(\Sigma, E)\in\max(\Sigma, E)\in\mathrm{Sch}(\Sigma)$ that finds to each algebraic specification (Σ, E) a scheme $\max(\Sigma, E)$ such that the following proposition is valid.

Proposition. *If* $F(\Sigma, E)$ *exists and for some pair of closed terms* t_1^s, t_2^s $F(\Sigma, E)\models t_1^s \neq t_2^s$ *then* $(\Sigma, E + t_1^s \neq t_2^s + \mathrm{gen}(\Sigma) + \max(\Sigma, E))\to F(\Sigma, E)$.

$\mathrm{Max}(\Sigma, E)$ is a somewhat complex scheme which says of an algebra that it is impossible to impose on it any nontrivial E-congruence. The scheme variables of $\max(\Sigma, E)$ are one binary predicate $\varphi_s(x^s, y^s)$ for each sort $s\in S$ $(= sorts(\Sigma))$; φ_s will represent the s-component of a candidate congruence (on A). $\max(\Sigma, E)$ is composed of three subschemes:

$$\mathrm{congruence}(\vec{\phi}_s),\quad \mathrm{trivial}(\vec{\phi}_s)\quad \text{and}\quad e\text{-correct}(\vec{\phi}_s)\quad \text{for } e\in E.$$

Here: $\mathrm{congruence}(\vec{\phi}_s) \equiv \bigwedge_{s\in S}[\phi_s(x^s, x^s)\wedge\phi_s(x^s, y^s)\leftrightarrow\phi_s(y^s, x^s)]$

$$\wedge(\phi_s(x^s, y^s)\wedge\phi_s(y^s, z^s)\to\phi_s(x^s, z^s))$$

$$\wedge \bigwedge_{f: s_1\times\ldots\times s_k\to s}\left[\bigwedge_{i=1}^{k}\phi_{s_i}(x_i^s, y_i^s)\to\phi_s(f(x_1,\ldots,x_k), f(y_1,\ldots,y_k))\right]$$

$$\mathrm{trivial}(\vec{\phi}_s) \equiv (\bigwedge_{s\in S}\forall x^s y^s\,\phi_s(x^s, y^s)) \vee \bigwedge_{s\in S}\forall x^s y^s\,\phi_s(x^s, y^s)\leftrightarrow x^s = y^s$$

$$e\text{-correct}(\vec{\phi}_s) \equiv \forall\vec{x}\left[\bigwedge_{i=1}^{n}\phi_{s_i}(t_i^{s_i}(\vec{x}), r_i^{s_i}(\vec{x}))\to\phi_s(t^s(\vec{x}), r^s(\vec{x}))\right]$$

with $\vec{x}$ the set of free variables of e:

$$\bigwedge t_i^{s_i} = r_i^{s_i} \to t^s = r^s.$$

Finally $\max(\Sigma, E)$ is composed:

$$\max(\Sigma, E) \equiv \text{congruence}(\phi_s) \wedge \bigwedge_{e \in E} e\text{-correct}(\phi_s) \to \text{trivial}(\phi_s).$$

1.6.3. DSL/SMS *and initial algebra semantics.* The next step is to consider initial algebra semantics. There is an important subcase to do with computable algebras.

Proposition. *If* $T(\Sigma, E) \cong F(\Sigma, E)$ *and for some pair* t_1^s, t_2^s *of closed* Σ*-terms* $T(\Sigma, E) \models t_1^s \neq t_2^s$, *then*

$$(\Sigma, E + t_1^s \neq t_2^s + \text{gen}(\Sigma)) \to T(\Sigma, E).$$

Proof. Suppose that the initial and final algebra semantics of (Σ, E) coincide (up to isomorphism), and that $T(\Sigma, E) \models t_1^s \neq t_2^s$.

Then certainly $T(\Sigma, E) \models_s E + t_1^s \neq t_2^s + \text{gen}(\Sigma)$. Moreover, assume $A \models_s E + t_1^s \neq t_2^s + \text{gen}(\Sigma)$, then A is minimal, whence one may choose a congruence $\equiv$ with $A \cong T_{\Sigma/\equiv}$. Because $A \models E$, $\equiv$ extends $\equiv_E$. Further, as $\equiv$ is nontrivial, it is included in $\equiv_E^L$. Now $\equiv_E = \equiv_E^L$, thus $\equiv = \equiv_E$ and $A \cong T_{\Sigma/\equiv} = T_{\Sigma/\equiv_E} = T(\Sigma, E)$.

It was shown in Bergstra and Tucker [4] that each computable algebra A can be specified (using auxiliary functions) with an algebraic specification (Σ', E') such that $T(\Sigma', E') \cong F(\Sigma', E')$. This state of affairs implies that the above proposition is by no means vacuous. A kind of converse is also here; if D denotes a finite set of inequalities of closed Σ-terms, and (Σ, E) is a finite algebraic specification, then:

Proposition. *For any* $A \in \text{Alg}(\Sigma)$, $(\Sigma, E + D + \text{gen}(\Sigma)) \to A$ *implies that* A *is a computable algebra.*

Proof. We derive two implications for closed terms t_1, t_2

$$A \models t_1^s = t_2^s \Rightarrow (\Sigma, E) \vdash t_1^s = t_2^s$$
$$A \models t_1^s \neq t_2^s \Rightarrow (\Sigma, E + D) \vdash t_1^s \neq t_2^s$$

As $\vdash$ is recursively enumerable and both conclusions are mutually exclusive these implications yield an effective decision procedure for $A \models t_1^s = t_2^s$, the existence of which is a sufficient criterion for A being computable.

Both implications have similar proofs, if suffices to look at no. 1., and to derive $A \models t_1^s \neq t_2^s$ from $(\Sigma, E \cup D) \not\vdash t_1^s = t_2^s$. Indeed if $(\Sigma, E \cup D) \not\vdash t_1^s = t_2^s$ the completeness theorem for first order logic implies the existence of a Σ-algebra B with $B \models E \cup D$ and $B \models t_1^s \neq t_2^s$. Let C be the minimal Σ-subalgebra of B. Then $C \in \text{Alg}_s(\Sigma, \mathbb{F})$, thus, using the assumption of this proposition $C \cong A$. Consequently $A \models t_1^s \neq t_2^s$.

Finally we hit upon the question: how to describe initial algebra semantics in DSL/SMS in a natural manner? At present we see *no solution to this issue*

other than using initial algebra semantics on top of SMS. The DSL specification $(\Sigma, E+D+\text{gen}(\Sigma))$ suffices for quite some proof theory and if one wishes a unique initial algebra $T(\Sigma,E)=I\,\text{Mod}_s(\Sigma, E+D+\text{gen}(\Sigma))$ can be singled out by some appropriate operator I.

2. Parametrized Data Types

2.1. Semantics of Parametrized Data Types

In this section we outline an initial segment of a theory of parametrized data types as it could look like from the point of view of DSL/SMS. For conventional theory on this topic we refer to [5–7, 12, 14].

A parametrized data type is given as a mapping

$$\alpha\colon \text{Mod}_s(\Sigma,\mathbb{F})\to\text{Mod}(\Delta)$$

(with Δ a signature extending Σ). This α must, moreover, satisfy three fundamental requirements:

(ii) $\alpha(A)$ is an expansion of A,

(ii) $\alpha(A)$ is generated from $|A|$ by Δ,

(iii) if $A\cong A'$ then $\alpha(A)\cong\alpha(A')$

(where $A, A'\in\text{Mod}_s(\Sigma,\mathbb{F})$ of course).

Having so defined parametrized data types it is useful to have a notion of isomorphism amongst p.d.t.'s available. We write $\alpha\cong\alpha'$ if both are mappings: $\text{Mod}_s(\Sigma,\mathbb{F})\to\text{Mod}(\Delta)$ and for all $A\in\text{Mod}_s(\Sigma,\mathbb{F})\ \alpha(A)\cong\alpha'(A)$.

This fixes the semantics of parametrized data types mathematically speaking.

2.2. Specifications of Parametrized Data Types

A specification for a parametrized data type is a notation $(\Sigma,\mathbb{F})\to(\Sigma',\mathbb{F}')_\Delta$. If $\Delta=\Sigma'$ it will be omitted.

This specification is well-defined if the following is true:

(i) $(\Sigma,\mathbb{F})\subseteq(\Sigma',\mathbb{F}')$,

(ii) for each $A\in\text{Mod}_s(\Sigma,\mathbb{F})$ there is an expansion A' of A in $\text{Mod}_s(\Sigma',\mathbb{F}')$ and this expansion A' is unique up to isomorphism,

(iii) $\Sigma\subseteq\Delta\subseteq\Sigma'$.

Suppose that $(\Sigma,\mathbb{F})\to(\Sigma',\mathbb{F})_\Delta$ is well-defined. Given a p.d.t. $\alpha\colon \text{Mod}_s(\Sigma,\mathbb{F})\to\text{Mod}(\Delta)$ we say that $(\Sigma,\mathbb{F})\to(\Sigma',\mathbb{F})_\Delta$ specifies α if for each $A\in\text{Mod}_s(\Sigma,\mathbb{F})$ there is an expansion A' of A in $\text{Mod}_s(\Sigma',\mathbb{F})$ such that $A'/\Delta\cong\alpha(A)$.

Thus a specification may involve auxiliary (hidden) functions and sorts. Clearly, if $\alpha\cong\alpha'$ then a specification for α is also a specification for α'.

2.3. Examples of Parametrized Data Types

2.3.1. $(\Sigma_A, \mathbb{F}_{FA}) \to (\Sigma_{SOA}, \mathbb{F}_{SOA})$ specifies the transition of A to 'sets of A'. See notations in 1.5.1, 1.5.6.

2.3.2. Augmenting bounded counters with addition. Let $(\Sigma_{CO}, \mathbb{F}_{BCO})$ be as in 1.3.3. Add to Σ_{CO} a function $+ : C \times C \to C$ thus obtaining Σ_{COA} and obtain $\mathbb{F}_{BCOA}$ as $\mathbb{F}_{BCO} \cup \{x+0=x, x+S(y)=S(x \cdot y)\}$. Then $(\Sigma_{CO}, \mathbb{F}_{BCO}) \to (\Sigma_{COA}, \mathbb{F}_{BCOA})$ specifies the enrichment of a bounded counter with addition.

2.3.3. Adding a cardinality function to the system of finite subsets of A. Take $(\Sigma_{SOA}, \mathbb{F}_{SOA})$ as in 1.5.6. Its standard models are just two-sorted structures with one being a domain, the other one containing all finite subsets of the domain.

Now let $\Sigma_{SOA\#}$ be the enrichment of Σ_{SOA} with I and a function $\# : \mathrm{SOA} \to I$. Take

$$\mathbb{F}_{SOA\#} = \mathbb{F}_{\omega} \cup \mathbb{F}_{SOA} \cup \{\#(\emptyset)=0, \mathrm{INS}(x,X) \neq X \to \# \mathrm{INS}(x,X) = S(\#(X))\}.$$

Then $(\Sigma_{SOA}, \mathbb{F}_{SOA}) \to (\Sigma_{SOA\#}, \mathbb{F}_{SOA\#})$ specifies the enrichment with integers and a cardinality function of sets of A.

3. WPL: A Program Construction for Local Data Types

3.1. Syntax

The optimal description of a program language involves an alphabet and a few inductive clauses that construct composed programs from simple ones. The **while** programs are a nice example of this. The system to be explained here requires some bookkeeping of both syntactic and semantic nature, in order to formulate conditions under which program constructions are allowed.

The specific feature of this program notation is that it allows for the local introduction of extra functions and sorts. This is easily possible due to the flexible specification of parametrized data types in DSL/SMS. As such WPL is an application of our theory. Further many interesting program equivalences result by eliminating local types from programs; this opens an area for experimenting with program equivalence proofs.

The following constructions serve as inductive clauses in a simultaneous definition of: WPL$(\Sigma, \mathbb{F})$, the WPL programs over $(\Sigma, \mathbb{F})$ and FV(T) the free variables of T. Below the precise mechanism is formulated through points (i), ..., (v).

$x_i^s := t^s$	assignment
$T_1 ; T_2$	composition
while b **do** T **od**	iteration
if b **then** T_1 **else** T_2 **fi**	conditional branch
new $x_i^s := t^s$; T **end**	new variable block
local $(\Sigma', \mathbb{F}')$ **on** $(\Sigma, \mathbb{F})$; T **end**	local type block

(i) if $s \in \mathbf{sorts}(\Sigma)$, $t^s \in \mathrm{Ter}_s(\Sigma)$ then $x_i^s := t^s \in \mathrm{WPL}(\Sigma, \mathbb{F})$, for each $\mathbb{F} \subseteq \mathrm{Sch}(\Sigma)$.

$$\mathrm{FV}(x_i^s := t^s) = \mathrm{FV}(t^s) \cup \{x_i^s\}.$$

(ii) if $T_i \in \mathrm{WPL}(\Sigma_i, \mathbb{F}_i)$, $i = 1, 2$ then $T_1 ; T_2 \in \mathrm{WPL}(\Sigma_1 \cup \Sigma_2, \mathbb{F}_1 \cup \mathbb{F}_2)$:

$$\mathrm{FV}(T_1 ; T_2) = \mathrm{FV}(T_1) \cup \mathrm{FV}(T_2).$$

(iii) similarly, if b is a quantifier free assertion in $\mathrm{Ass}(\Sigma)$ and $T \in \mathrm{WPL}(\Sigma, \mathbb{F})$,

$$\mathrm{FV}(\textbf{while } b \textbf{ do } T \textbf{ od}) = \mathrm{FV}(b) \cup \mathrm{FV}(T).$$

(iv) similar to (ii) and (iii) for the conditional branch,

(v) if $T \in \mathrm{WPL}(\Sigma, \mathbb{F})$, $s \in \mathbf{sorts}(\Sigma)$, $t^s \in \mathrm{Ter}_s(\Sigma)$ and $x_i^s \notin \mathrm{FV}(t^s)$ then **new** $x_i^s := t^s$; T **end** $\in \mathrm{WPL}(\Sigma, \mathbb{F})$ and its free variables are $\mathrm{FV}(T) - \{x_i^s\}$,

(vi) for the local type block construct to be applicable it is required that:

(1) $T \in \mathrm{WPL}(\Sigma', \mathbb{F}')$,

(2) $\mathrm{FV}(T)$ contains no variables for sorts in $\mathbf{sorts}(\Sigma') - \mathbf{sorts}(\Sigma)$,

(3) $(\Sigma, \mathbb{F}) \to (\Sigma', \mathbb{F}')$ is a well-defined parametrized data type specification.

The resulting program is in $\mathrm{WPL}(\Sigma, \mathbb{F})$ and its free variables coincide with those of T.

3.2. *Semantics*

The semantics of WPL will be defined 'pointwise', per program, specification and structure, in an operational way. Assume as data for this definition to be given the following:

(i) a specification $(\Sigma, \mathbb{F})$,

(ii) a program $T \in \mathrm{WPL}(\Sigma, \mathbb{F})$,

(iii) a structure $A \in \mathrm{Mod}_s(\Sigma, \mathbb{F})$,

(iv) a listing $(x_{i_1}^{s_1}, \ldots, x_{i_k}^{s_k})$ of variables, including the variables in $\mathrm{FV}(T)$,

(v) σ: a notation for $(s_1, i_1, \ldots, s_k, i_k)$.

Given these data there is a function

$$\mathscr{M}_A^\sigma(T) : A_{s_1} \times \ldots \times A_{s_k} \to A_{s_1} \times \ldots \times A_{s_k}.$$

The definition of the $\mathscr{M}_A(T)$ follows an inductive pattern along the construction of T. Except for the block rules the clauses for this induction are entirely standard and not repeated here.

The semantics of the new variable block is defined according the dynamic scope, and the local type block construct requires some detailed comments.

Let σ be a listing for T' with

$$T' = \textbf{local}(\Sigma', \mathbb{F}') \textbf{ on } (\Sigma, \mathbb{F}) ; \ T \textbf{ end} \in \mathrm{WPL}(\Sigma, \mathbb{F}).$$

Given $A \in \mathrm{Mod}_s(\Sigma, \mathbb{F})$ find an expansion A^* of A to a structure in $\mathrm{Mod}_s(\Sigma', \mathbb{F}')$. Because σ is a listing for T as well $\mathscr{M}_A^\sigma(T)$ is well-defined. Take

$$\mathscr{M}_A^\sigma(T') = \mathscr{M}_{A^*}^\sigma(T).$$

It is not hard to see that $\mathcal{M}_A^o(T')$ defined this way is independent of a particular choice of A^* (all choices are isomorphic).

3.3. Examples

3.3.1. Addition of integers

```
T: z:=x;
   new h:=0;
       while h≠y
       do z:=S(z)
          h:=S(h)
       od
   end
```

$T \in \mathrm{WPL}(\Sigma_\omega, \mathbb{F}_\omega)$. T adds x and y placing the result in z and leaves x, y unaffected.

With $(\Sigma_{\omega/+/.}, \mathbb{F}_{\omega/+/.})$ as in 1.5.2. observe that

$$(\Sigma_\omega, \mathbb{F}_\omega) \rightarrow (\Sigma_{\omega/+/.}, \mathbb{F}_{\omega/+/.})$$

is a well-defined specification. The following program T' is equivalent to T:

```
T': local (Σ_{ω/+/.}, F_{ω/+/.}) on (Σ_ω, F_ω);
       z:=x+y
    end
```

3.3.2. List searching. The datatype LOA, lists over A is given by a specification $(\Sigma_{\mathrm{LOA}}, \mathbb{F}_{\mathrm{LOA}})$.

Σ_{LOA}: **sorts**: I, A, LOA.
functions: $S: I \rightarrow I$, $\mathrm{VAL}: \mathrm{LOA} \times I \rightarrow A$
constants: $a \in A$, $0 \in I$.

$\mathbb{F}_{\mathrm{LOA}}$: $\mathbb{F}_\omega \cup \{\forall z^{\mathrm{LOA}}, w^{\mathrm{LOA}} (\forall x^{\mathrm{I}}\, \mathrm{VAL}(z^{\mathrm{LOA}}, x^{\mathrm{I}}) = \mathrm{VAL}(w^{\mathrm{LOA}}, x^{\mathrm{I}}) \rightarrow z^{\mathrm{LOA}} = w^{\mathrm{LOA}})\}$.

$\mathrm{Mod}_s(\Sigma_{\mathrm{LOA}}, \mathbb{F}_{\mathrm{LOA}})$ contains all three-sorted structures $(\omega, A, \mathrm{LOA}, S, \mathrm{VAL}, 0, a)$ with $(\omega, s, 0)$ isomorphic to the integers, A a nonempty set and LOA a collection of functions: $\omega \rightarrow A$, with $\mathrm{VAL}(\alpha, x) = \alpha(x)$ for $\alpha \in A$.

We are interested in a program $T \in \mathrm{WPL}(\Sigma_{\mathrm{LOA}}, \mathbb{F}_{\mathrm{LOA}})$ with free variable x^{I} and X^{LOA} that should compute the following partial function $\psi: \mathrm{I} \times \mathrm{LOA} \rightarrow \mathrm{I} \times \mathrm{LOA}$; $\psi(x, X) = (x', X)$ with x' the smallest $i \in \omega$ such that for some $j \in \omega$ $j < i$ $\mathrm{VAL}(X, i) = \mathrm{VAL}(X, j)$.

The straightforward solution is:

```
new z:=0; x:=S(0);
    while VAL(X,z)≠VAL(X,x)
    do z:=0; x:=S(x);
       while VAL(X,z)≠VAL(X,x) ∧ x≠S(z)
       do z:=S(z)
       od
    od
end
```

Another program, using a locally defined datatype is much simpler. Let $(\Sigma_{LSOA}, \mathbb{F}_{LSOA}) = (\Sigma_{LOA}, \mathbb{F}_{LOA}) \cup (\Sigma_{SOA}, \mathbb{F}_{SOA})$ with SOA as in Example 1.5.6. Then this program also works:

local $(\Sigma_{LSOA}, F_{LSOA})$ **on** $(\Sigma_{LOA}, \mathbb{F}_{LOA})$
  new $Z := \emptyset$;
    $x := 0$;
    while $\neg$ELT(VAL$(X, x), Z)$
    do $Z :=$ INS(VAL$(X, x), Z)$
      $x := S(x)$
    od
  end
end

References

1. Bergstra, J.A., Klop, J.W.: A formalized proof system for total correctness of while programs. In: The proceedings of ISI Conf., SLNCS 137, 1982
2. Bergstra, J.A., Klop, J.W.: Proving program inclusion using Hoare's Logic. Mathematical Centre Department of Computer Science Research Report IW 176, Amsterdam, 1981
3. Bergstra, J.A., Terlouw, J.: A characterization of program equivalence in terms of Hoare's Logic. In: Springer LNCS 123, 1981
4. Bergstra, J.A., Tucker, J.V.: The completeness of the algebra specification methods for datatypes. Mathematical Centre Department of Computer Science Research Report IW 165 Amsterdam, 1981
5. Ehrich, H.-D.: On the theory of specification, implementation and parametrization of abstract datatypes. Research report Dortmund, 1978
6. Ehrig, H.E., Kreowski, H.-J., Thatcher, J.W., Wagner, E.G., Wright, J.B.: Parametrized datatypes in algebraic specification languages. Proc. 7th. ICALP SLNCS 85, 1980
7. Ganzinger, H.: Parametrized specifications: parameter passing and optimizing implementation. Report TUM-18110, Technische Universität München, August 1981
8. Goguen, J.A., Thatcher, J.W., Wright, J.B.: Abstract datatypes as initial algebras and correctness of data representations. In: Proceedings of ACM Conference on Computer Graphics, Pattern Recognition and Data Structure, ACM, New York, 1975
9. Guttag, J.V.: The specification and application to programming of abstract datatypes. Ph.D. Thesis. University of Toronto. Department of Computer Science, 1975
10. Kamin, S.: Final datatype specifications: a new datatype specification method. Proc. 7th ACM Symp. Principles Progr. Lang., Las Vegas, 1980
11. Kaphengst, H., Reichel, H.: Algebraische Algorithmentheorie. VEB Robotron, Dresden WIB, 1971
12. Thatcher, J.W., Wagner, E.G., Wright, J.B.: Datatype specification: parametrization and the power of specification techniques. Proc. SIGACT 10th. Annual STOC, 1978
13. Wand, M.: Final algebra semantics and datatype extensions. JCSS **19**, 27-44 (1979)
14. Wirsing, M.: An analysis of semantic models for algebraic specifications. Lecture Notes for the International Summer School on theoretical foundations of programming methodology, München 1981

Received April 29, 1982 October 15, 1982