First Order Data Types and First Order Logic

by

Ralf Treinen

A 01/91

Saarbrücken, Januar 1991

Abstract: This paper concerns the relation between parameterized first order data types and first order logic. Augmenting first order logic by data type definitions yields in general a strictly stronger logic than first order logic. We consider some properties of the new logic for fixed data type definitions. While our new logic always fulfills the downward Skolem-Löwenheim property, compactness is fulfilled if and only if for the given data type definition the new logic has the same expressive power than first order logic. We show that this last property is undecidable.

Ralf Treinen, Fachbereich 14 - Informatik, Im Stadtwald, Universität des Saarlandes, W6600 Saarbrücken, Germany, `treinen@cs.uni-sb.de`

A short version appeared in T. Ito and A. R. Meyer, eds., *Theoretical Aspects of Computer Software*, Sendai, Japan, September 1991, pages 594–614, Springer LNCS vol. 526.

Contents

1 Introduction

The use of modules for data abstraction is now a well-established principle in software design, see for instance [Bis86]. From the programmers point of view a module is a piece of encapsulated software that propagates only a well-defined subset of its data structures and operations to its environment, we call this the *export* part of the module. Outside the module these data structures and operations are accessible only via their names, the implementation remains hidden from the users of the module. The module may use data structures and operations defined elsewhere in the program, this leads to the important concept of parameterization: The *parameter* part of module specifies the sorts and operations that have to be supplied to the module. In the following we will always consider modules as parameterized modules, even if the the parameter part is not stated explicitly in the syntax. This interpretation of modules reflects in the semantics: The semantics of module now has to be defined as a function that maps the denotations of the paramter part to the denotations of the export part.

Among the modularization concepts known from programming languages we here only mention the clusters of CLU ([LG86], [LAB*81]), the packages of Ada[1] ([DoD81], [ANSI83]), the modules of MODULA-2([Wir85]) and the structures of ML ([Mac86]). Generic data types as in ML or Miranda[2] ([Tur85], [Tur86]) provide another parameterization concept by abstracting basic sorts from data type definitions. Furthermore parameterization comes naturally with specification languages, no matter whether they are operational, axiomatic (algebraic) or algorithmic ([EM90]). A non-exhaustive list of specification languages using parameterization is Alphard ([WLS76], [Sha81]), CLEAR([BG80]), ACT ONE ([EM85]) and OBSCURE ([LL87], [LL90]). All these languages differ substantially in the methods used for implementing, resp. specifying, data structures and operations. In this paper we take an abstract approach and present an idealized language for expressing modules. The language follows the Algorithmic Specification language of [Loe87] and can be seen as a programming language as well as a specification language. We only consider modules defining data structures and operations, higher order functions are not included in our language. We have a purely functional point of view, that is modules construct new "exported" algebras from given "parameter" algebras.

We do not distinguish between the import- and the parameter part of a module as it is done in [EM85]: In our sense the import part comprises all the sorts and operations that may be used by a module but are not defined in it. [EM85] call this the import part while their parameter part designates some sorts and operations that are common to the import and the export part. We choose the name "parameter" in order to emphasize that the semantics of a module depends exactly on the meaning of these sort and operation symbols.

In our idealized language functions are defined by general recursive programs as in functional programming languages, but without syntactic sugar as for instance the let-construct or patern matching. Data structures are defined by constructor functions, this kind of data definition is known from languages like ML, Miranda or Algorithmic Specifications. The Algorithmic

[1] Ada is a registerd trademark of the United States Departement of Defense.

[2] Miranda is a registered trademark of Research Software Ltd.

Specification method furthermore introduces subset and quotient operations on algebras, but as shown in [Loe87] these are not relevant for the logic and we are allowed to drop them here. Although restricted in expressive power we claim that our idealized language provides a representation of interesting subsets of the languages above (also the imperative ones in the case of absence of global variables, such that the functional perspective is retained).

This idealized language can be seen as a generalization of program schemes (see for instance [Gre75]). In fact, when we restrict the module language by excluding the definition of new sorts we meet exactly the situation of recursive program schemes, where for some module a parameter algebra corresponds to an interpretation of a program scheme. While in the theory of program schemes one is interested in deriving properties of schemes that hold for all interpretations from some fixed class, we are here interested in another question.

Our interest in modules is motivated by their use in the top-down design of software. Given some properties of the export algebra of a module, we would like to know the exact requirements to the parameter algebra that ensure that the properties are satisfied. This leads to the central notion of a weakest parameter condition: For a given module m, a formula v over the parameter signature of m is called a *weakest parameter condition* of a formula w over the export signature of m if for each parameter algebra P: P fulfills v iff the semantics of m, applied to P, fulfills w.

In order to investigate the existence of weakest parameter conditions we incorporate the semantics of modules into a new logic that extends first order logic. In this new logic the models are parameter algebras and the formulas are first order formulas over the export signature of some given module. We always refer to some fixed module, that is the modules themselves do not occur in the formulas of our new logic. In the terminology of [KT90], our logic is an endogenous logic about modules, not an exogenous one.

The Logic of Effective Definitions (EDL) of [Tiu81] considers completely unstructered schemes that are a generalization of recursive program schemes. Model theoretic and proof theoretic properties of EDL are discussed in [Tiu81]. In contrast to EDL our logic incorporates data structures defined by constructors. Furthermore we concentrate on the relation between first order logic and the new logic as described above.

This paper is organized as follows: In the next section we first shortly review the notions we use in the rest of the paper. In Section 3 we define syntax and semantics of our module language and show some basic properties of the semantics. Section 4 defines the central notions with regard to the logic. The fundamental model-theoretic properties of our new logic are investigated in Section 5. Section 6 addresses decidability questions.

2 Preliminaries

The purpose of this section is fixing the notations used in this paper, not giving complete definitions.

We summarize some basic notions about signatures and algebras. A complete set of definitions is given in [EM85].

A *signature* is a pair (S, F), where

- S is a set of sort symbols.
- F is a set of S-sorted function symbols, that is each $f \in F$ is equipped with an arity $arity(f) \in S^* \times S$. For $arity(f) = (s_1, \ldots, s_n, s)$ we will often write $f : s_1, \ldots, s_n \to s$.

If $\Sigma = (S, F)$ is a signature and F' is a S-sorted set of function symbols we write $\Sigma \cup F'$ for $(S, F \cup F')$. The intersection of signatures $\Sigma_i = (S_i, F_i)$, $i = 1, 2$, is defined as $\Sigma_1 \cap \Sigma_2 := (S_1 \cap S_2, F_1 \cap F_2)$. A signature $\Sigma_1 = (S_1, F_1)$ is a *subsignature* of a signature $\Sigma_2 = (S_2, F_2)$ if $S_1 \subseteq S_2$ and $F_1 \subseteq F_2$. A *variable family* for a signature $\Sigma = (S, F)$ is a family $X = (X_s)_{s \in S}$ of sets of variable symbols that are pairwise disjoint and disjoint to F. We will often use set notation instead of the exact family notation. For a variable family $X = (X_s)_{s \in S}$ and (S, F)-algebra A $\Gamma_{X,A}$ or shortly Γ_A denotes the set of A-assignments.

If X is a variable family for the signature $\Sigma = (S, F)$, $T_\Sigma(X)$ is the set of terms built with F and X. The set of ground terms $T_\Sigma(\emptyset)$ is also written as T_Σ. For $t \in T_\Sigma(X)$ the sort of t $sort(t) \in S$ and the set of free variables of t $free(t) \subseteq X$ are defined as usual, $T_{\Sigma,s}(X)$ denotes the subset of terms with sort s.

Let $\Sigma = (S, F)$ be a signature. A Σ-algebra A consists of

- a set s^A for each $s \in S$, called the carrier set of sort s, and
- a function $f^A : s_1^A, \ldots, s_n^A \to s^A$ for each function symbol $f \in F$ with $f : s_1, \ldots, s_n \to s$

Let $\Sigma = (S, F)$ be a signature. The Σ-algebra B is a Σ-*subalgebra* of the Σ-algebra A (notation: $B \subseteq A$), if:

- for all $s \in S : s^B \subseteq s^A$
- for all $f \in F$, $f\colon s_1, \ldots, s_n \to s : f^B = f^A |_{s_1^B \times \ldots \times s_n^B}$

If Σ_1 is a subsignature of Σ_2 and A a Σ_2-algebra then $A |_{\Sigma_1}$ denotes the restriction of A to Σ_1. A Σ-algebra A induces an interpretation function for terms:

$$A\colon T_\Sigma(X) \to (\Gamma_{X,A} \to A)$$

We assume from the reader basic knowledge on first order logic (see for instance [End72], [CK90]). We only consider first order logic with equality as the only predicate symbol, therefore we can consider algebras as models in the sense of first order logic. We write $A, \alpha \models w$ if the formula w is satisfied by the model A and the assignment $\alpha \in \Gamma_A$, if w is a sentence we write $A \models w$ in this case. This generalizes to sets W of sentences by $A \models W$ if $A \models w$ for all $w \in W$. For a class $\mathcal{C}$ of algebras we write $\mathcal{C} \models w$ if $A \models w$ for each $A \in \mathcal{C}$.

We use a somewhat sloppy notation for extensions of algebras adopted from [CK90]. If A is a (S, F)-algebra, $(S \cup \{c\}, F)$ an extension of (S, F) by one constant symbol c of sort s and $a \in s^A$, then we denote by (A, a) the $(S \cup \{c\}, F)$-algebra that coincides with A on (S, F) and assignes a to c.

$\supset\!\!\!\subset$ is the equivalence junctor. For a sentence w and terms t_1, t_2 of the same sort where t_1 does not have a bound variable occurrence in w, $w(t_1/t_2)$ denotes the sentence obtained by substituting every occurrence of t_1 in w by t_2. It is understood that bound variables are renamed such that no free variable of t_2 is captured by a quantifier of w.

A Σ-algebra A is an *elementary submodel* of a Σ-algebra B if $A \subseteq B$ and for all Σ-formulas w and all A-assignments $\alpha \in \Gamma_A \subseteq \Gamma_B$: $A, \alpha \models w$ iff $B, \alpha \models w$. This is a stronger notion than beeing a submodel, especially elementary submodels have the same first order theory ([CK90]).

We use some basic notions from the theory of the semantics of programs, see for instance [LS87].

3 Modules

3.1 Syntax

The syntax of modules as given in this section closely resembles the syntax of Algorithmic Specifications as given in [Loe87].

Before we give the exact syntax of a module we define the notion of a *standard signature*. A signature will be called standard if some distinguished sort and operation symbols are present in it.

Definition 1 *A signature (S, F) is called a* standard signature ([Loe87]) *if:*

- *S contains the sort bool and*
- *F contains the following function symbols:*

$$\begin{array}{rcll}
true & : & \to bool & \\
false & : & \to bool & \\
\bot_s & : & \to s & \textit{for all } s \in S \\
\texttt{ifthenelse}_s & : & bool, s, s \to s & \textit{for all } s \in S \\
=_s & : & s, s \to bool & \textit{for all } s \in S
\end{array}$$

In [Loe87] $\bot_s$ and $\texttt{ifthenelse}_s$ have not been included in the standard signatures.

Informally speaking, a module as it will be formally defined in Definition 2 consists of the following items (see Figure 1 for an example):

- the signature of the parameter algebra
- the body of the module containing
  - the list of newly defined sorts. For each new sort s we implicitely define functions $\perp_s$, $=_s$ and $\texttt{ifthenelse}_s$ of appropriate sort.
  - the list of constructor symbols that will be used to define the carriers of the new sorts. The domain sorts of these constructors may include sorts from the parameter algebra, their range must of course be a new sort. The introduction of a constructor, say c, automatically entails the definition of a pertaining test function $\texttt{is}_c\texttt{?}$ and selector functions $\texttt{select}_c^j$. This bears some similarities with the list data structure of Common LISP ([Ste90]) where `nil` and `cons` are constructors, `consp` is a test predicate and `car` and `cons` serve as selectors[3].
  - the list of recursive function symbols. Together with all the functions mentioned above they build the set of accessible function symbols.
  - a recursive program for the recursive function symbols. This recursive program may make use of all accessible function symbols.
- the set of exported function symbols that designates some subset of the set of accessible function symbols as exported. The other function symbols remain hidden inside the module.

We do not consider hiding of sorts here since this is not relevant from the logical point of view.

Definition 2 *A* module *is a tuple*

$$(PS_m, PF_m, NS_m, K_m, NF_m, EF_m, PR_m)$$

where PS_m, PF_m, NS_m, K_m and NF_m are pairwise disjoint sets and

- *(PS_m, PF_m) is a standard signature, called Σ_{Pm}. PS_m contains the* parameter sorts *of m.*
- *$(PS_m \cup NS_m, PF_m \cup K_m \cup NF_m)$ is a signature and the range of all function symbols in K_m is an element of NS_m. Extending this signature by the following set of function symbols:*

$$\begin{array}{rl} & \{\texttt{ifthenelse}_s\colon bool, s, s \to s \mid s \in NS_m\} \\ \cup & \{=_s\colon s, s \to bool \mid s \in NS_m\} \\ \cup & \{\texttt{is}_c\texttt{?}\colon s \to bool \mid s \in NS_m\} \\ \cup & \{\texttt{select}_c^j\colon s \to s_j \mid c \in K_m\ ,\ c\colon s_1, \ldots, s_j, \ldots, s_n \to s\} \\ \cup & \{\perp_s\colon \to s \mid s \in NS_m\} \end{array}$$

  we obtain a standard signature $\Sigma_{Am} = (AS_m, AF_m)$.

[3]This is of course no complete analogy since LISP does not obey a strong typing discipline.

PAR	**SORTS**	*elem*
	OPNS	$0\colon \to elem$
		$+\colon elem, elem \to elem$
BODY	**SORTS**	*list*
	CONS	$nil\colon \to list$
		$cons\colon elem, list \to list$
	FCTS	$app\colon list, list \to list$
		$sum\colon list \to elem$
	PROG	$app(l_1, l_2) \Leftarrow \mathtt{if}\ \mathtt{is}_{nil}\mathtt{?}(l_1)\ \mathtt{then}\ l_2$ $\mathtt{else}$ $cons(\mathtt{select}^1_{cons}(l_1), app(\mathtt{select}^2_{cons}(l_1), l_2))$
		$sum(l) \Leftarrow \mathtt{if}\ \mathtt{is}_{nil}\mathtt{?}(l)\ \mathtt{then}\ 0$ $\mathtt{else}\ \mathtt{select}^1_{cons}(l) + sum(\mathtt{select}^2_{cons}(l))$

Figure 1: An example of a module definition.

- PR_m *is a recursive program of the form*

$$\begin{pmatrix} f_1(x_{1,1}, \ldots, x_{1,l_1}) & \Leftarrow & t_1 \\ \vdots & & \vdots \\ f_n(x_{n,1}, \ldots, x_{n,l_n}) & \Leftarrow & t_n \end{pmatrix}$$

  where $NF_m = \{f_1, \ldots, f_n\}$*, for all* i *the* $x_{i,j}$ *are pairwise distinct and of appropriate sort and* $t_i \in T_{\Sigma_{Am}}(\{x_{i,1}, \ldots, x_{i,l_i}\})$ *of appropriate sort.*

- EF_m *is a subset of* AF_m *such that* $\Sigma_{Em} := (AS_m, EF_m)$ *is a standard signature.*

The pair $(\Sigma_{Pm}, \Sigma_{Em})$ *constitutes the* signature of the module m*.*

Figure 1 contains an example of a module written in a more user friendly syntax. We will be somewhat sloppy in syntax and will not mention the standard parts of the signatures, drop the sort indices if known from the context and allow mixfix syntax if convenient. Furthermore we will not mention EF_m if identical to AF_m.

In order to formulate Theorem 2 we will need the notion of an extension of a module by a set of constants.

Definition 3 *Let* m *be a module and* C *a set of constant symbols disjoint from all components of* m*. Then the* extension of m by C *is the module*

$$(PS_m, PF_m \cup C, NS_m, K_m, NF_m, EF_m, PR_m)$$

3.2 Semantics

The semantics of modules as defined in this section again resembles [Loe87]. In contrast to [Loe87] where the semantics is defined denotationally we here take an approach that is adopted from the algebraic semantics method ([Gue79]).

An algebra over a standard signature will be called standard if it assignes the intended meanings to the standard parts of the signature.

Definition 4 *Let $\Sigma = (S, F)$ be a standard signature. A Σ-algebra A is called a* standard algebra ([Loe87]) *if*

- $bool^A = \{\underline{\text{true}}, \underline{\text{false}}, \bot_{bool}\}$ *and* $true^A = \underline{\text{true}}$, $false^A = \underline{\text{false}}$, $\bot^A_{bool} = \bot_{bool}$
- *for all $s \in S$ and $x_1, x_2 \in s^A$:*

$$\begin{array}{lcl} \texttt{ifthenelse}^A_s(\underline{\text{true}}, x_1, x_2) & = & x_1 \\ \texttt{ifthenelse}^A_s(\underline{\text{false}}, x_1, x_2) & = & x_2 \\ \texttt{ifthenelse}^A_s(\bot_{bool}, x_1, x_2) & = & \bot^A_s \end{array}$$

- *For each function symbol $f \in F$ the denotation f^A is continuous with respect to the following ordering $\sqsubseteq_s$:*

$$x_1 \sqsubseteq_s x_2 \quad \text{iff} \quad x_1 = x_2 \quad \text{or} \quad x_1 = \bot^A_s$$

Alg_Σ denotes the class of all standard algebras with signature Σ.

Note that, since a standard signature contains a constant symbol $\bot_s$ of each sort s, a standard algebra always contains a distinguished carrier $\bot^A_s$ of each sort s.

In the following we will always consider standard algebras. We will now in several steps define the semantics of a module m. The semantics will be formalized as a function $\mathcal{M}$ that maps a module m with signature (Σ_P, Σ_E) and a Σ_P-standard algebra A to a Σ_E-standard algebra B.

First we define the carrier sets of the algebra constructed by the semantics of a module.

Definition 5 *Let m be a module with signature (Σ_P, Σ_E) and $A \in Alg_{\Sigma_P}$. We define a family of sets $(D^{m,A}_s)_{s \in AS_m}$ as follows:*

- *for each $s \in PS_m$ let $\bar{D}^{m,A}_s := s^A - \{\bot^A_s\}$*
- *for all $s \in NS_m$ define sets $\bar{D}^{m,A}_s$ by simultaneous induction:*
  if $k \in K_m$, $k\colon s_1, \ldots, s_n \to s$ and $d_i \in \bar{D}^{m,A}_{s_i}$ for $i = 1, \ldots, n$
  then $k(d_1, \ldots, d_n) \in \bar{D}^{m,A}_s$
- *now define for all $s \in AS_m$: $D^{m,A}_s := \bar{D}^{m,A}_s \cup \{\bot_s\}$*

Now we define an intermediate algebra that extends the parameter algebra by the newly defined sorts and operations except the recursive functions. This intermediate algebra will then be used in order to define the semantics of the recursive functions and to obtain the complete semantics of the module.

Definition 6 *Let m be a module with signature (Σ_P, Σ_E) and $A \in Alg_{\Sigma_P}$. We define an algebra A^* with signature $(AS_m, AF_m \setminus NF_m)$ as follows:*

- $A^* |_{\Sigma_P} := A$
- $s^{A^*} := D_s^{m,A}$ *for all* $s \in NS_m$
- $\perp_s^A := \perp_s$ *for all* $s \in NS_m$
- *For all* $s \in NS_m$ $\mathtt{ifthenelse}_s$ *and* $=_s$ *obtain their meaning according to the definition of standard algebra*
- *For all* $k\colon s_1, \ldots, s_n \to s \in K_m$*:*

$$k^{A^*}(x_1, \ldots, x_n) := \begin{cases} k(x_1, \ldots, x_n) & \textit{if } x_i \neq \perp_{s_i}^A \textit{ for all } i \\ \perp_s^A & \textit{otherwise} \end{cases}$$

- *For all* $c\colon s_1, \ldots, s_n \to s \in K_m$*:*

$$\mathtt{is}_c\mathtt{?}^{A^*}(x) := \begin{cases} \underline{\text{true}} & \textit{if } x = c(x_1, \ldots, x_n) \textit{ for some } x_i \\ \perp_{bool} & \textit{if } x = \perp_s^A \\ \underline{\text{false}} & \textit{otherwise} \end{cases}$$

- *For all* $c\colon s_1, \ldots, s_n \to s \in K_m$*:*

$$\mathtt{select}_c^{j\,A^*}(x) := \begin{cases} x_j & \textit{if } x = c(x_1, \ldots, x_j, \ldots, x_n) \\ & \quad\quad \textit{for some } x_i \neq \perp_{s_i}^A, i = 1 \ldots n \\ \perp_{s_j}^A & \textit{otherwise} \end{cases}$$

Lemma 1 *A^* is a standard algebra.*

Proof: This follows from the continuity of the sequential $if - then - else$ function ([LS87]) and the strictness of the remaining functions. □

In order to define the semantics of the recursive function symbols we need the notion of iteration. Intuitively, the n-times iteration of a term t is obtained by n-times simultaneously unfolding all occurrences of recursive function symbols in t. The remaining recursive function symbols are replaced by $\perp$, this yields a term that is assigned a meaning by the algebra A^*.

Definition 7 *Let m be a module with signature (Σ_P, Σ_E) and $t \in T_{\Sigma_{E_m}}(X)$. Then for each $n \in \mathbb{N}$, $t\langle n\rangle$ is the term obtained by n-fold application of the full substitution computation rule on t and then replacing each occurrence of recursive function symbols by $\perp$.*

In the terminology of [Gue79], Definition 3.22, this is the n-th element t^n of the Kleene sequence of t.

The reader is referred to [Gue79] for a formal definition.

We are now ready to give the complete definition of the semantics of a module. Our definition of the semantics of recursive function symbols is in the spirit of algebraic semantics (see for instance [Gue79], in contrast to [Loe87] where a denotational approach was token). The advantage of the algebraic semantics is that it makes the distinction between the recursion structure given by the program and the interpretation of the base functions explicit. Furthermore we will make use of the iterations of a term later in the logic.

Definition 8 *Let m be a module with signature (Σ_P, Σ_E). Then $\mathcal{M}(m)$ is a function*

$$\mathcal{M}(m)\colon Alg_{\Sigma_P} \to Alg_{\Sigma_E}$$

where for all $A \in Alg_{\Sigma_P}$:

- $\mathcal{M}(m)(A) \mid_{(AS_m, EF_m \setminus NF_m)} := A^* \mid_{(AS_m, EF_m \setminus NF_m)}$
- *for all $f : s_1, \ldots, s_n \to s \in NF_m \cap EF_m$, $a_i \in s_i^{A^*}$:*

$$f^{\mathcal{M}(m)(A)}(a_1, \ldots, a_n) := \bigsqcup_{i \geq 0} A^*(f(x_1, \ldots, x_n)\langle i\rangle)(\alpha[x_i \leftarrow a_i])$$

  for an arbitrary assignment $\alpha \in \Gamma_{A^*}$.

The choice of the assignment α is arbitrary since $x_1, \ldots x_n$ are the only free variables in the terms under consideration. The existence of the least upper bound of this set of values is a simple consequence of the fact that all cpo's are flat.

Lemma 2 *$\mathcal{M}(m)(A)$ is a standard algebra.*

Proof: follows from Lemma 1 and the fact that the denotations of recursive function symbols are again continuous, see for instance [LS87]. □

Lemma 3 *Let m be a module with signature (Σ_P, Σ_E), $A \in Alg_{\Sigma_P}$, $B = \mathcal{M}(m)(A)$, $f\colon s_1, \ldots, s_n \to s \in EF \cap NF_m$ and $a_i \in s^B$. Let $\alpha \in \Gamma_B$ with $\alpha(x_i) = a_i$ for all $i = 1, \ldots, n$. Then*

- *either $f^B(a_1, \ldots, a_n) = \perp_s^B$ and $B(f(x_1, \ldots, x_n)\langle j\rangle)(\alpha) = \perp_s^B$ for all $j \in \mathbb{N}$*
- *or $f^B(a_1, \ldots, a_n) = c \neq \perp_s^B$ and there is a j_0 such that for all $j \geq j_0$: $B(f(x_1, \ldots, x_n)\langle j\rangle)(\alpha) = c$*

Proof: This is a simple consequence of the continuity of the functions and of the fact that the cpo is flat, see [LS87]. □

$$
\begin{tikzcd}
A \arrow[rr, shift left, "\text{apply } \mathcal{M}(m)"] \arrow[d, dash, "\subseteq"] & & \mathcal{M}(m)(A) \arrow[ll, shift left, "\text{restrict to } \Sigma_P"] \arrow[d, dash, "\subseteq"] \arrow[dr, dash, "\subseteq"] & \\
B \arrow[rrr, shift right, "\text{apply } \mathcal{M}(m)"'] & & B' \arrow[ll, "\text{restrict to } \Sigma_P"'] & \mathcal{M}(m)(B) = B''
\end{tikzcd}
$$

Figure 2: The algebras used in Lemma 6

3.3 Basic Properties of the Semantics

Our semantics obeys the *persistency* condition. Intuitively this means that the sorts and operations of the parameter algebra are not modified by the semantics of a module.

Lemma 4 *Let m be a module with signature (Σ_P, Σ_E) and $A \in Alg_{\Sigma_P}$. Then*

$$\mathcal{M}(m)(A) \mid_{\Sigma_P \cap \Sigma_E} = A \mid_{\Sigma_P \cap \Sigma_E}$$

We now show two lemmas that we will need for the proof of Theorem 1.

Lemma 5 *Let m be a module with signature (Σ_P, Σ_E), $A, B \in Alg_{\Sigma_P}$ and $A \subseteq B$. Then $\mathcal{M}(m)(A) \subseteq \mathcal{M}(m)(B)$*

Proof: The inclusion of the carrier sets is obvious. The coincidence of the semantics of the new function symbols is easily shown from the definitions. □

Lemma 6 *Let m be a module with signature (Σ_P, Σ_E), $\Sigma_P \subseteq \Sigma_E$, $A \in Alg_{\Sigma_P}$ and $B' \in Alg_{\Sigma_E}$ with $B' \subseteq \mathcal{M}(m)(A)$.*
Then $B' = \mathcal{M}(m)(B' \mid_{\Sigma_P})$.

Proof: (See also Figure 2.) Define $B := B' \mid_{\Sigma_P}$ and $B'' := \mathcal{M}(m)(B)$. By the definition of subalgebra we get immediately $B \subseteq A$. We now show that B' and B'' have the same carrier sets. For the carrier sets of parameter sort this is immediate by Lemma 4.

$s^{B''} \subseteq s^{B'}$ for $s \in NS$:

This follows by structural induction from $B \subseteq A$ since B' is closed under the denotation of the constructors.

$s^{B'} \subseteq s^{B''}$ for $s \in NS$:

Assume there is some carrier of B' that is not a carrier of B''. Let x be a minimal carrier (w.r.t. to the subterm ordering) of B' that is not a carrier of B''. By construction x must be of the form $c(x_1, \ldots, x_n)$ for some constructor c. Since B' is closed under the denotation of the selectors, $x_1, ..., x_n$ are carriers of B' and by minimality of x are also carriers of B''. Therefore $x = c(x_1, \ldots, x_n)$ is also a carrier of B'', this contradicts the assumption.

Since B' and B'' have the same carriers and are both subalgebras of the same superalgebra, the denotations of the function symbols also coincide, so $B' = B''$. $\square$

4 Logic

In this section we show how to apply first order logic to modules.

4.1 Basic Definitions and Properties

Definition 9 *For a standard signature $\Sigma = (S, F)$ let WFF_Σ be the set of first order sentences over the language $\langle S, F, \{=_s \mid s \in S\}\rangle$ where each $=_s$ is a binary predicate symbol.*

If m is a module with signature (Σ_P, Σ_E) we denote WFF_{Σ_P} by $PWFF_m$ and WFF_{Σ_E} by WFF_m.

The symbol $=_s$ is overloaded here: we use it as a function symbol with arity $s, s \to \mathit{bool}$ and as a binary predicate symbol of the logic. Again we will drop the sort index if convenient.

Let us emphasize that we only consider sentences, that is first order formulas without free variables. By definition the set of first order sentences, considered as a subset of the set of first order formulas, is generated from the atoms (here: equalities) by negation, conjunction and existential quantification, but we also use the other usual logical junctors as syntactic abbreviations (see [End72] for a complete set of definitions). Furthermore we use the following:

$$
\begin{array}{rcl}
(t_1 \sqsubseteq t_2) & \text{stands for} & (t_1 = \bot \vee t_1 = t_2) \\
\forall x \in s \,.\, w & \text{stands for} & \forall x : s \,.\, x = \bot_s \vee w \\
\exists x \in s \,.\, w & \text{stands for} & \exists x : s \,.\, x \neq \bot_s \wedge w
\end{array}
$$

We now come to the central definition of this paper. For a sentence $w \in WFF_\Sigma$ and (not necessarily standard) algebra A we write as usual $A \models w$ if A is a model of w, see again [End72] for complete definitions. The point is that we can now use the sematics of a module with signature (Σ_P, Σ_E) in order to express properties of standard Σ_P-algebras by Σ_E-sentences.

Definition 10 *Let m be a module with signature (Σ_P, Σ_E), $A \in Alg_{\Sigma_P}$ and $w \in WFF_m$. We define*

$$A \,]\!\models_m w \qquad \Leftrightarrow \qquad \mathcal{M}(m)(A) \models w$$

For $W \subseteq WFF_m$ we write $A\,]\!\models_m W$ if $A\,]\!\models_m w$ for all $w \in W$. $]\!\models_m w$ means $A\,]\!\models_m w$ for all $A \in Alg_{\Sigma_P}$. Furthermore $Th_m(A) := \{w \in WFF_m \mid A\,]\!\models_m w\}$

For example let m be the module of Figure 1 and Nat the extension of the algebra of natural numbers to a standard algebra. Then

$$Nat\,]\!\models_m \forall l_1, l_2 : \mathit{list} \,.\, \mathit{sum}(\mathit{app}(l_1, l_2)) = \mathit{sum}(l_1) + \mathit{sum}(l_2)$$

As an immediate consequence of the persistency of the semantics (Lemma 4) we get

Lemma 7 *Let m be a module with signature (Σ_P, Σ_E), $A \in Alg_{\Sigma_P}$ and $w \in WFF_{\Sigma_P \cap \Sigma_E}$. Then*

$$A \models w \qquad \textit{iff} \qquad A]\!\models_m w$$

This means that our new logic is at least as expressive as first order logic. Later we will see that, depending on the module under consideration, there is in general indeed a gain in expressiveness.

4.2 Classes of Parameter Algebras

We are not always interested in parameter algebras from the whole class Alg_{Σ_P}. Instead it is often natural to restrict the parameter algebras to some subclass of Alg_{Σ_P}. The choice of this subclass should depend only on the input signature. We put some reasonable constraints on the possible classes of parameter algebras that we will need in the following.

We call a class of algebras *compact* if the compactness theorem of first order logic holds in this class of models.

Definition 11 *A class $\mathcal{C}$ of Σ-algebras is called* compact *if for each set $W \subseteq WFF_\Sigma$ of formulas the following holds:*

> *If each finite subset of W has a model in $\mathcal{C}$*
> *then W has a model in $\mathcal{C}$*

The choice of a particular class of parameter algebras is formally expressed by the concept of a *domain operator*:

Definition 12 *A domain operator $\Im$ maps each standard signature Σ to a subclass $\Im_\Sigma$ of Alg_Σ such that the following holds:*

1. *$\Im_\Sigma$ is compact.*
2. *$\Im_\Sigma$ is closed under elementary submodels.*
3. *For any sort symbol s in Σ and constant symbol c not in Σ:*

$$\Im_{\Sigma \cup \{c:s\}} = \{(A, a) \mid A \in \Im_\Sigma \textit{ and } a \in s^A\}$$

The following mappings are no domain operators:

1. F mapping each signature to the class of all standard algebras where all functions terminate for all but a finite set of input values. Using the compactness theorem of first order logic it is easy to show that for non-trivial Σ F_Σ does not fulfill compactness.

(1) $\forall x{:}\, \mathbf{bool}\,.\,(x = \bot_{bool} \vee x = true \vee x = false)$
(2) $\bot_{bool} \neq true$
(3) $\bot_{bool} \neq false$
(4) $true \neq false$
(5) $\forall x, y{:}\, s\,.\,[(x =_s y) = true \supset\!\subset (x = y \wedge x \neq \bot_s)]$
(6) $\forall x, y{:}\, s\,.\,[(x =_s y) = false \supset\!\subset (x \neq y \wedge x \neq \bot_s \wedge y \neq \bot_s)]$
(7) $\forall x, y{:}\, s\,.\,[(x =_s y) = \bot_{bool} \supset\!\subset (x = \bot_s \vee y = \bot_s)]$
(8) $\forall x, y{:}\, s\,.\,\mathtt{if}\ true\ \mathtt{then}\ x\ \mathtt{else}\ y\ \mathtt{fi} = x$
(9) $\forall x, y{:}\, s\,.\,\mathtt{if}\ false\ \mathtt{then}\ x\ \mathtt{else}\ y\ \mathtt{fi} = y$
(10) $\forall x, y{:}\, s\,.\,\mathtt{if}\ \bot_{bool}\ \mathtt{then}\ x\ \mathtt{else}\ y\ \mathtt{fi} = \bot_s$
(11) $\forall \vec{x}_i, \vec{y}_i{:}\, s\,.\,[\bigwedge_{i=1\ldots n}(x_i \sqsubseteq y_i) \supset f(x_1, \ldots, x_n) \sqsubseteq f(y_1, \ldots, y_n)]$

Figure 3: Axiom schemes for standard algebras.

2. The operator mapping each signature to the class of standard algebras with cardinality greater than $\aleph_0$ violates the closure under elementary submodels. This is an easy consequence of the sharpened Skolem-Löwenheim Theorem of first order logic.

3. There is an imported case not covered by the definition of a domain operator: The mapping TG that carries each signature Σ to the class of term-generated models is no domain operator since the last condition of Definition 12 is violated.

On the other hand the next lemma shows that a wide class of mappings satisfies the constraints of Definition 12:

Lemma 8 *Let ι be a mapping that maps each standard signature $\Sigma = (S, F)$ to some set of formulas $\iota(\Sigma) \subseteq WFF_\Sigma$ where only constants from $\{true, false\} \cup \{\bot_s \mid s \in S\}$ are allowed. Then the operator mapping each signature Σ to the class of standard algebras that are models of $\iota(\Sigma)$ is a domain operator.*

Proof: First observe that the class of standard algebras is exactely the class of models of the set of axioms given by the axiom schemes of Figure 3 where s varies over all sorts and f varies over all function symbols of the signature. Therefore the compactness property and the closure under elementary submodels are easy consequences of the pertaining theorems of first order logic: The compactness theorem (Theorem 1.3.22 in [CK90]), respectively the sharpened Skolem-Löwenheim Theorem (Theorem 3.1.6 in [CK90]). The proof of the third constraint is trivial. □

As a consequence the following operators are indeed domain operators:

1. The operator $\Im^f$ mapping each signature to the full class of standard algebras.

2. The operator $\Im^{strict}$ mapping each signature to the class of standard algebras where all functions except $if - then - else$ are strict.

3. The operator mapping each signature to the class of standard algebras where all functions are sequential ([Vui74]).

On the other hand the operator TG mentioned above is not of great interest in this framework, since we do not want to require that all functions of an algebra are explicitelly listed in the parameter part of a module. For instance we could define a module "list of elements" that we want to apply to several algebras without worrying about all the other functions that might be present in the parameter algebra.

We therefore claim that the constraints in Definition 12 are reasonable. Note that we did not require closure of the domain operator under the semantics of modules, although this would be an acceptable constraint in view of vertical composition of modules. To be precise, we do *not* require that $\mathcal{M}(m)(A) \in \Im_{\Sigma_E}$ for $A \in \Im_{\Sigma_P}$.

4.3 Parameter Conditions

The notion of a parameter condition links first order logic to our new logic.

Definition 13 *Let m be a module with signature (Σ_P, Σ_E), $\Im$ a class operator and $w \in WFF_m$. A sentence $v \in PWFF_m$ is a $\Im, m$-*parameter condition *of w if for all $A \in \Im_{\Sigma_P}$:*

$$A \models v \qquad \Rightarrow \qquad A]\models_m w$$

*A sentence $v \in PWFF_m$ is a $\Im, m$-*weakest parameter condition *of w if for all $A \in \Im_{\Sigma_P}$:*

$$A \models v \qquad \Leftrightarrow \qquad A]\models_m w$$

The following lemma is immediate by the definition:

Lemma 9 *Weakest parameter conditions are unique up to equivalence, that is: let m be a module with signature (Σ_P, Σ_E), $\Im$ a domain operator and $w \in WFF_m$. Then for all weakest parameter conditions $v_1, v_2 \in PWFF_m$: $\Im_{\Sigma_P} \models v_1 \Leftrightarrow v_2$*

We illustrate the important notion of a weakest parameter condition with some examples.

1. Let m be the module of Figure 1. The formula

$$\begin{array}{l}(\forall x \in elem\,.\, 0 + x = x \wedge x + 0 = x) \wedge \\ (\forall l_1, l_2 \in list\,.\, sum(app(l_1, l_2)) = sum(l_1) + sum(l_2))\end{array}$$

   has the $\Im^{strict}, m$-weakest parameter condition:

$$\begin{array}{l}(\forall x \in elem\,.\, 0 + x = x \wedge x + 0 = x) \wedge \\ (\forall x_1, x_2, x_3 \in elem\,.\, x_1 + (x_2 + x_3) = (x_1 + x_2) + x_3)\end{array}$$

PAR	**SORTS**	*elem*
	OPNS	$0\colon \to elem$
		$pred\colon elem \to elem$
BODY	**FCTS**	$isstandard\colon elem \to bool$
	PROG	$isstandard(x) \Leftarrow \texttt{if } x = 0 \texttt{ then } true$
		$\qquad\qquad\qquad\qquad \texttt{else } isstandard(pred(x))$

Figure 4: A module used for distinguishing standard from nonstandard models.

PAR	**SORTS**	*elem*
BODY	**SORTS**	*list*
	CONS	$nil\colon \to list$
		$cons\colon elem, list \to list$
	FCTS	$isin\colon elem, list \to bool$
	PROG	$isin(e,l) \Leftarrow \texttt{if is}_{nil}\texttt{?}(l) \texttt{ then } false$
		$\qquad \texttt{else if select}^1_{cons}(l) = e \texttt{ then } true$
		$\qquad\qquad \texttt{else } isin(e, \texttt{select}^2_{cons}(l))$

Figure 5: A module used for distinguishing finite algebras from infinite ones.

---

2. Consider the module m of Figure 4. The formula

$$w := \forall x \in elem.isstandard(x) = true$$

does not have a $\Im^{strict}$, m-weakest parameter condition since the standard and the nonstandard model of arithmetic have the same first order theory while the former fulfills w and the latter does not.

3. The last example shows that the same problem might also occur with primitive recursive functions. Take the module m of Figure 5. The class of standard algebras that fulfill the formula

$$w := \exists l \in list\,.\,\forall x \in elem\,.\,isin(x,l) = true$$

is exactely the class of finite Σ_{P_m}-algebras. Since the class of finite algebras cannot be described by means of first order logic ([CK90]) w does not have a $\Im^f$, m-weakest parameter condition.

These last examples show that our new logic is more expressive than first order logic. In the next section we will discuss the model theoretic properties of the new logic that reflect this gain in expressiveness.

The following lemma gives a special case in which a $\Im$, m-weakest parameter condition always exists:

Lemma 10 *Let m be a module with $NF_m = \emptyset$ and $\Im$ a domain operator. Then for each $w \in WFF_m$ there exists a $\Im, m$-weakest parameter condition. Furthermore for each formula w the weakest parameter condition is computable.*

Proof: The proof follows from procedures for solving equational problems in term algebras ([CL89], [Mah88]). See [Tre91] and [Buh91] for details how to apply these results to the semantics of modules. □

5 Properties of the New Logic

We now consider two basic model-theoretic properties of our new logic. Lindström ([Lin69]) has shown that first order logic is the only logic fulfilling countable compactness and the Skolem-Löwenheim property (see also [Mon76], [CK90]). Besides the fact that he considers logical systems with the whole class of algebras as domain (instead of standard algebras in our case) his theorem applies in our case only to an endogenous variant of our logic where all possible modules are considered. Here we are interested in obtaining theorems about the logical properties of distinguished modules.

5.1 Downward Skolem-Löwenheim

In this subsection we show that our new logic has the downward Skolem-Löwenhein property. An analogous result has been proven in [Tiu81] for the Logic of Effective Definitions by translation to the logic $L\omega_1\omega$ ([Kei71]). The proof of the theorem below directly depends on the closure of the domain operators under elementary submodels. For domain operators described by first order axioms this closure property follows from the strong version of the first order Skolem-Löwenheim property, infinitary logic is not needed here.

Theorem 1 *Let m be a module with signature (Σ_P, Σ_E), $\Im$ a domain operator and $A \in \Im_{\Sigma_P}$. Then for each family $Z = (Z_s)_{s \in PS}$ of sets with cardinality at most $\aleph_0$ and $Z_s \subseteq s^A$ there is a $B \in \Im_{\Sigma_P}$ of cardinality $\aleph_0$ that contains Z such that $\mathcal{M}(m)(B)$ is an elementary submodel of $\mathcal{M}(m)(A)$.*

Proof: Without loss of generality let Z contain $\aleph_0$ many elements. Furthermore we may assume $\Sigma_P \subseteq \Sigma_E$ since elementary submodels are invariant under restriction of the signature.

By the sharpened downward Skolem Löwenheim Theorem of first order logic (Theorem 3.1.6 of [CK90]) there is an elementary submodel B' of $\mathcal{M}(m)(A)$ with cardinality $\aleph_0$ containing Z. Let $B := B' |_{\Sigma_P}$. B contains Z and therefore has cardinality $\aleph_0$. By Theorem 6: $B' = \mathcal{M}(m)(B)$. Since B' is an elementary submodel of $\mathcal{M}(m)(A)$, B is also an elementary submodel of A. By the closure of $\Im_{\Sigma_P}$ under elmentary submodels $B \in \Im_{\Sigma_P}$. □

Corollary 1 *Let m be a module with signature (Σ_P, Σ_E) and $A \in \Im_{\Sigma_P}$ of infinite cardinality. Then there is a $B \in \Im_{\Sigma_P}$ of cardinality $\aleph_0$ with $Th_m(A) = Th_m(B)$.*

Proof: Let B be the model according to Theorem 1. By the properties of elementary submodels $\mathcal{M}(m)(A)$ and $\mathcal{M}(m)(B)$ have the same first order theory and so $Th_m(A) = Th_m(B)$. $\square$

5.2 Compactness

From first order logic it is known that the most applications of the compactness theorem require the introduction of new constant symbols in some intermediate step. These new constants in some sense allow to express an existential quantification over an inifinite conjunction of formulas. Therefore the compactness theorem can be used in order to show that a theory has a model containing an element satisfying some infinite set of formulas (see for instance Proposition 2.2.7 in [CK90]). In order to argue about compactness properties of our logic we therefore have to consider extensions of given modules, since including them into the parameter part is the only way to incorporate new constant symbols.

Definition 14 *Let m be a module with signature (Σ_P, Σ_E) and $\Im$ a domain operator. We say that $]\!\models_m$ is* $\Im$-compact *if for each $W \subseteq WFF_m$ the following holds:*

> *If for each finite $F \subseteq W$ there is a $A \in \Im_{\Sigma_P}$ with $A]\!\models_m F$, then there exists $B \in \Im_{\Sigma_P}$ with $B]\!\models_m W$*

Theorem 2 *Let m be a module and $\Im$ a domain operator. Then the following statements are equivalent:*

1. *For each extension m' of m, $]\!\models_{m'}$ is $\Im$-compact.*
2. *For each extension m' of m and $w \in WFF_{m'}$ w has a $\Im, m'$-weakest parameter condition*

Proof:

$\underline{(1) \Longleftarrow (2)}$

This is an easy consequence of the definition of a weakest parameter condition and of the compactness property of the domain operator $\Im$.

$\underline{(1) \Longrightarrow (2)}$

Assume that for each extension m' of m $]\!\models_{m'}$ is compact. We define the set W as the set of all formulas that belong to some arbitrary extension of m. Strictly speaking this is a set only if we fix some set of possible constant symbols, but we do not bother about set theoretic peculiarities here.

$$W := \bigcup_{m' \text{ extends } m} WFF_{m'}$$

For each $w \in W$ define

- $\phi_1(w)$ is the number of occurrences of existential quantifiers in w ranging over some new sort

- $\phi_2(w)$ is the number of occurrences of existential quantifiers in w ranging over some parameter sort plus the number of occurrences of $\neg$, $\wedge$ in w.

With the help of these notions we define a relation $\sqsubseteq$ on W by

$$w_1 \sqsubseteq w_2 :\Longleftrightarrow (\phi_1(w_1), \phi_2(w_1)) \leq_{\text{lex}} (\phi_1(w_2), \phi_2(w_2))$$

where $\leq_{\text{lex}}$ is the lexicographic extension of the ordering $\leq$ on natural numbers. From the properties of lexicographic orderings it is obvious that $\sqsubseteq$ is a well founded quasi ordering ([Der87]).

Now let $w \in W$ be a minimal formula with respect to $\sqsubseteq$ such that there exists an extension m' of m with $w \in WFF_{m'}$ and w does not have a weakest $\Im, m'$-parameter condition. Σ' denotes the parameter signature of m'. First we show that w must be an atomic formula. Note that for a given formula $v \in W$ we can restrict our attention to the minimal extension m^* of m such that $v \in WFF_{m^*}$. The addition of further constants does not affect the existence of a weakest parameter condition. We say that $v \in W$ has a weakest parameter condition (without mentioning the module) if it has a $\Im, m^*$-weakest parameter condition where m^* is the extension of m by the constants occurring in v.

1. Suppose $w = \exists x : s \,.\, v$ where s is a parameter sort. Let c be a new constant symbol not occurring in m' or w. By the minimality condition $v(x/c)$ has a weakest parameter condition r. We obtain a contradiction by showing that $\exists y : s \,.\, r(c/y)$ is a weakest parameter condition of w where y does not occur freely in r.

   Let m'' denote the extension of m' by $\{c\!: s\}$ and $A \in \Im_{\Sigma'}$.

	$A \models \exists y : s \,.\, r(c/y)$	
$\Leftrightarrow$	$(A, a) \models r$	for some extension (A, a) of A
$\Leftrightarrow$	$(A, a)]\!\models_{m''} v(x/c)$	since r is a $\Im, m''$ weakest parameter condition of $v(x/c)$
$\Leftrightarrow$	$(A, a)]\!\models_{m''} \exists x : s \,.\, v$	since c does not occur in m'' or w
$\Leftrightarrow$	$A]\!\models_{m'} \exists x : s \,.\, v$	since c does not occur in m' or w

2. Suppose $w = \exists x : s \,.\, v(x)$ where s is a new sort. Define

   $$C_s := T_{K,s}(X_{par}) \cup \{\perp_s\}$$

   where X_{par} is the familiy of variables of parameter sort. From the definition of the semantics it is immediate that for each $A \in \Im_{\Sigma'}$ and $a \in s^{\mathcal{M}(m')(A)}$ there is a $t \in C_s$ and an assignment $\alpha \in \Gamma_A$ with

   $$\mathcal{M}(m')(A)(t)(\alpha) = a \qquad (1)$$

For each finite set $F \subseteq C_s$, the formula

$$\bigvee_{t \in F} \exists free(t) \,.\, v(x/t)$$

has by minimality of w a $\Im, m'$-weakest parameter condition r_F which is itself a $\Im, m'$-parameter condition of w. Since w by assumption does not have a weakest parameter condition, for each finite $F \subseteq C_s$ there is a $A \in \Im_{\Sigma'}$ with

$$A\,]\!\models_{m'} \{\exists x : s \,.\, v\} \cup \{\forall free(t) \,.\, \neg v(x/t) \mid t \in F\}$$

Since $]\!\models_{m'}$ is compact there is an $A \in \Im_{\Sigma'}$ with

$$A\,]\!\models_{m'} \{\exists x : s \,.\, v\} \cup \{\forall free(t) \,.\, \neg v(x/t) \mid t \in C_s\}$$

This contradicts (1).

3. Suppose $w = \neg v$. By minimality of w v has a weakest parameter condition r. Then $\neg r$ must be a weakest parameter condition of w.

4. Suppose $w = v_1 \vee v_2$. By minimality of w v_1 and v_2 have weakest parameter conditions r_1 and r_2, respectively. Then $r_1 \vee r_2$ must be a weakest parameter condition of w.

We now know that w must be of the form $t_1 = t_2$. This formula is equivalent to

$$\underbrace{(t_1 = t_2 \wedge t_1 \neq \bot)}_{v_1} \vee \neg(\underbrace{t_1 \neq \bot}_{v2} \vee \underbrace{t_2 \neq \bot}_{v_3})$$

As in the cases (3),(4) above it follows that at least one of v_1, v_2, v_3 does not have a weakest parameter condition. Without loss of generality we assume that v_1 does not have a weakest parameter condition.

By Lemma 10 we know that for each natural number n there is a $\Im, m'$-weakest parameter condition r_n of $t_1\langle n\rangle = t_2\langle n\rangle \wedge t_1\langle n\rangle \neq \bot$. Observe that $]\!\models_{m'} \neg r_n \supset \neg r_m$ for $n > m$. Since r_n is a parameter condition for v_1 and since v_1 by assumption does not have a weakest parameter condition, for each finite set F of natural numbers there is a $A \in \Im_{\Sigma'}$ with

$$A\,]\!\models_{m'} \{v_1\} \cup \{\neg r_n \mid n \in F\}$$

By the compactness property of $]\!\models_{m'}$ there is a $A \in \Im_{\Sigma'}$ with

$$A\,]\!\models_{m'} \{t_1 = t_2 \wedge t_1 \neq \bot\} \cup \{t_1\langle n\rangle \neq t_2\langle n\rangle \vee t_1\langle n\rangle = \bot \mid n > 0\}$$

According to the properties of monotonic functions there are two possibilities:

- For all n: $\mathcal{M}(m')(A)(t_1\langle n\rangle = \bot$. This contradicts $\mathcal{M}(m')(A)(t_1) \neq \bot$ by Lemma 3.
- There is a n_0 such that $\mathcal{M}(m')(A)(t_1\langle n_0\rangle) \neq \bot$. Then for all $n > n_0$ $\mathcal{M}(m')(A)(t_1\langle n\rangle) \neq \bot$ and therefore $\mathcal{M}(m')(t_1\langle n\rangle) \neq \mathcal{M}(m')(t_2\langle n\rangle)$. This contradicts $\mathcal{M}(m')(A)(t_1) = \mathcal{M}(m')(A)(t_2)$ by Lemma 3.

$\square$

6 Decidability Questions

We now show that the existence of weakest parameter conditions is in general undecidable, even if the module does not introduce new sorts.

In order to show undecidability of the existence of weakest parameter conditions we have to take care that the domain operator under consideration is rich enough. If the domain operator is too trivial a weakest parameter condition always exists. We illustrate this remark with one example:

Take the domain operator $\Im_n$ that carries each signature to the class of finite standard algebras with cardinality less or equal to the fixed number n. In any $\Im_{n\Sigma}$ there exist up to isomorphism only finitely many algebras and each isomorphism class can be characterized by an appropriate formula. As in Lemma 8 we obtain that $\Im_n$ is indeed a domain operator. Note that the axiomatization of the isomorphism classes does involve the constant symbols, nevertheless the constraint on the constants is obviously fulfilled. On the other hand there is a weakest $\Im_n$,m-parameter condition for each formula w, namely the disjunction of those axioms associated to the isomorphism classes that satisfy w.

Therefore we require the domain operator to be non-trivial. In order to define non-triviality we use some notions from [WPP*83]:

Definition 15 *A domain operator* $\Im$ *is called* non-trivial *if for each hierarchical type* $T = (\Sigma, E, P)$ *where*

- P *is the specification BOOL*
- E *is a finite set of* Σ*-equations*
- T *is hierarchy-persistent*

the extension of the initial model of T *to a standard algebra is contained in* $\Im_\Sigma$*.*

The extension of A to a standard algebra *is obtained by extending the signature to a standard signature, assigning* $\perp$*,* `ifthenelse` *and* $=$ *their standard meaning and extending all functions of* A *strictly.*

The hierarchy-persistence here means that the equations of E do not "destroy" the datatype BOOL and do not introduce new elements of sort bool.

We use a result about two-head automata that turned out to be useful for undecidability results in the field of program schemes. The reason for the adequacy for program schemes is that no particular data types are required except bit sequences (these can be simulated by predicates) and the states of the finite control (these are coded directly in the program).

We shortly repeat the definition of a two-head automaton and the pertaining undecidability result. Details can be found in [LPP70] and [Gre75], see also [Ros63]. Here we consider only automata over a fixed binary alphabet $\{0, 1\}$.

A *two-head automaton* (THA for short) is a tuple

$$(Q_1, Q_2, q_0, q_a, q_r, \delta)$$

where Q_1 and Q_2 are finite sets, Q_1, Q_2, $\{q_0\}$, $\{q_a\}$ and $\{q_r\}$ are pairwise disjoint sets of states and δ is a transition function

$$\delta\colon (Q_1 \cup Q_2 \cup \{q_0\}) \times \{0,1\} \to Q_1 \cup Q_2 \cup \{q_a, q_r\}$$

Such an automaton is given as input an *infinite* sequence over $\{0,1\}$. The automaton operates similar to a finite state automaton but now has two read-only heads moving independently forward over the tape. In order to determine the next state the input is taken from the first head (resp. second head) iff the actual state is a member of $Q_1 \cup \{q_0\}$ (resp. Q_2). Then the head from which the input has been taken moves forward to the next position. Note that for a given input tape there are three possibilities:

- The automaton *accepts* its input iff it eventually reaches q_a.
- The automaton *rejects* its input iff it eventually reaches q_r.
- The automaton *diverges* on its input if it never reaches q_a or q_r.

$\mathcal{L}_A$ denotes the set of inputs accepted by A, $\mathcal{D}_A$ the set of inputs on which A diverges. We use the following result

Lemma 11 ([LPP70]) *It is not semidecidable whether for a THA A*

- *the set $\mathcal{L}_A$ is empty.*
- *the set $\mathcal{D}_A$ is not empty.*

Sketch of the proof (see [LPP70], [Gre75] for details): For a given Turing machine T we can effectively construct a THA A_T such that the only inputs accepted by A_T are the tapes starting with a finite computation sequence of T with empty input, followed by some arbitrary sequence. Using this construction we can reduce the halting problem for Turing machines to the emptiness problem for THA's. □

The module of Figure 6 simulates a THA in the following sense. To a given Σ_P algebra B we associate the input tape $tape_B$ that is defined by

$$tape_B(i) := \begin{cases} 0 & \text{if } B(contents0(next^i(start))) = true \\ 1 & \text{if } B(contents0(next^i(start))) = false \end{cases}$$

Obviously each possible input tape is a $tape_B$ for some Σ_P algebra B.

PAR **SORTS** $tapeposition$
$value$
OPNS $start\colon \rightarrow tapeposition$
$next\colon tapeposition \rightarrow tapeposition$
$contents0\colon tapeposition \rightarrow bool$
$a\colon \rightarrow value$
$f\colon value \rightarrow value$
$test\colon value \rightarrow bool$

BODY **FCTS** $H\colon \rightarrow bool$
$F_q\colon tapeposition, tapeposition, value \rightarrow bool$
for all states q of the automaton

PROG

$$
\begin{array}{lll}
H & \Leftarrow & F_{q_0}(start, start, a) \\
F_q(p_1, p_2, x) & \Leftarrow & \texttt{if } contents0(p_1) \\
 & & \quad \texttt{then } F_{\delta(q,0)}(next(p_1), p_2, x) \\
 & & \quad \texttt{else } F_{\delta(q,1)}(next(p_1), p_2, x) \\
 & & \text{for all states } q \in Q_1 \\
F_q(p_1, p_2, x) & \Leftarrow & \texttt{if } contents0(p_2) \\
 & & \quad \texttt{then } F_{\delta(q,0)}(p_1, next(p_2), x) \\
 & & \quad \texttt{else } F_{\delta(q,1)}(p_1, next(p_2), x) \\
 & & \text{for all states } q \in Q_2 \\
F_r(p_1, p_2, x) & \Leftarrow & F_r(p_1, p_2, x) \\
F_a(p_1, p_2, x) & \Leftarrow & \texttt{if } test(x) \\
 & & \quad \texttt{then } true \\
 & & \quad \texttt{else } F_{q_0}(start, start, f(x))
\end{array}
$$

EXPORT $H\colon \rightarrow bool$

Figure 6: A module simulating a two-head automaton used for Theorem 3

Lemma 12 *Let A be a two-head automaton and m be the pertaining module according to Figure 6. For each $B \in Alg_{\Sigma_P}$ and $\alpha \in \Gamma_B$:*

$$
\begin{array}{lll}
tape_B \in \mathcal{L}_A & \Rightarrow & \mathcal{M}(m)(B)(F_{q_0}(start, start, x))(\alpha) = \\
 & & \mathcal{M}(m)(B)(\texttt{if}\, test(x)\, \texttt{then}\, true\, \texttt{else}\, F_{q_0}(start, start, f(x)))(\alpha) \\
tape_B \notin \mathcal{L}_A & \Rightarrow & \mathcal{M}(m)(B)(F_{q_0}(start, start, x))(\alpha) = \bot
\end{array}
$$

Proof: This follows easily from the definitions. $\square$

Lemma 13 *Let m be the module associated to the THA A according to Figure 6 and $\Im$ a non trivial domain operator.*

1. *If $\mathcal{L}_A = \emptyset$ then $]\models_m H = \bot$*
2. *If $\mathcal{L}_A \neq \emptyset$ then the formula $(H \neq \bot)$ does not have a $\Im, m$-weakest parameter condition.*

Proof: (1) follows immediately from Lemma 12. For part (2), let $t \in \mathcal{L}_A$ and n be the last position of t visited by any of the heads of A when feeded with input t. Suppose v is a $\Im, m$ weakest import condition of $(H \neq \bot)$.

We can decscribe the relevant part of t (that is the initial part of t up to position n) by a finite set of equations:

$$e_t := \bigwedge_{i=0\ldots n} \begin{cases} contents0(next^i(start)) = true & \text{if } t(i) = 0 \\ contents0(next^i(start)) = false & \text{if } t(i) = 1 \end{cases}$$

From Lemma 12 we conclude that for each $B \in Alg_{\Sigma_P}$ with $B \models e_t$:

$$B\,]\models_m H \neq \bot \;\Leftrightarrow\; B \models test(f^i(a)) = true \text{ for some } i \tag{2}$$

On the other hand each set of the form

$$\{v, e_t\} \cup \{test(f^i(a)) = false \mid i \leq n_0\}$$

has by non-triviality of $\Im$ a model in $\Im_{\Sigma_P}$, namely the extension of the initial model of $(BOOL, \Sigma_P, E)$ to a standard algebra where

$$\begin{aligned} E &= \{e_t\} \\ &\cup \{test(f^i(a)) = false \mid i \leq n_0\} \\ &\cup \{test(f^{n_0+1}(x)) = true\} \end{aligned}$$

By compactness of $\Im$ there is an algebra in $\Im_{\Sigma_P}$ satisfying

$$\{v, e_t\} \cup \{test(f^i(a)) = false \mid i \in \mathbb{N}\}$$

This contradicts (2). $\square$

As a consequence we can in each formula, if $\mathcal{L}_A = \emptyset$, replace H bei $\bot_{bool}$ thus obtaining a $\Im, m$-weakest parameter condition. Therefore we get the first undecidability result of this section:

Theorem 3 *For a non-trivial domain operator $\Im$ the following sets are not semidecidable:*

- *the set of modules m such that all formulas $w \in WFF_m$ have a $\Im, m$-weakest parameter condition*

PAR **SORTS** $tapeposition$
$value$
OPNS $start\colon \to tapeposition$
$next\colon tapeposition \to tapeposition$
$contents0\colon tapeposition \to bool$
$a\colon \to value$
$f\colon value \to value$
$test\colon value \to bool$

BODY **FCTS** $H\colon \to bool$
$F_q\colon tapeposition, tapeposition, value \to bool$
for all states q of the automaton

PROG $H \Leftarrow F_{q_0}(start, start, a)$
$F_q(p_1, p_2, x) \Leftarrow$ `if` $test(x)$ `then` $true$
`else if` $contents0(p_1)$
`then` $F_{\delta(q,0)}(next(p_1), p_2, f(x))$
`else` $F_{\delta(q,1)}(next(p_1), p_2, f(x))$
for all states $q \in Q_1$
$F_q(p_1, p_2, x) \Leftarrow$ `if` $test(x)$ `then` $true$
`else if` $contents0(p_2)$
`then` $F_{\delta(q,0)}(p_1, next(p_2), f(x))$
`else` $F_{\delta(q,1)}(p_1, next(p_2), f(x))$
for all states $q \in Q_2$
$F_r(p_1, p_2, x) \Leftarrow true$
$F_a(p_1, p_2, x) \Leftarrow true$

EXPORT $H\colon \to bool$

Figure 7: A module simulating a two-head automaton used for Theorem 4

- *the set of pairs* (w, m) *where* m *is a module,* $w \in WFF_m$ *and* w *has a* $\Im, m$*-weakest parameter condition*

In order to show that the sets of Theorem 3 are also not co-semidecidable we use again a reduction of a not semidecidable property of THA. The module of Figure 7 is in some sense a twisted version of the module presented in Figure 6. Now we let the function H terminate iff the input tape is rejected or accepted by the automaton, while a possible infinite sequence of tests is performed iff the automaton diverges on the input tape. The proof is analogous to the first proof, we therefore only state the key lemma and the concluding theorem:

Lemma 14 *Let m be the module associated to the THA A according to Figure 7 and $\Im$ a non trivial domain operator.*

1. *If $\mathcal{D}_A = \emptyset$ then $]\models_m H = true$*
2. *If $\mathcal{D}_A \neq \emptyset$ then the formula $(H \neq \bot)$ does not have a $\Im, m$ weakest parameter condition.*

Theorem 4 *For a non-trivial domain operator $\Im$ the following sets are not semidecidable:*

- *the set of modules m such that some formula $w \in WFF_m$ does not have a $\Im, m$ weakest parameter condition*
- *the set of pairs (w, m) where m is a module and $w \in WFF_m$ and w does not have a $\Im, m$ weakest parameter condition*

I wish to thank Thomas Lehmann, Joachim Philippi, and Jacques Loeckx and for comments and discussions.

References

[ANSI83] American National Standards Institute. *The Programming Language Ada Reference Manual. LNCS vol. 155*, Springer, 1983.

[BG80] R. M. Burstall and J. A. Goguen. Semantics of CLEAR, a specification language. In D. Björner, editor, *Abstract Softare Specifications*, pages 292–332, Springer LNCS, vol. 86, 1980.

[Bis86] Judy Bishop. *Data Abstraction in Programming Languages*. Addison–Wesley, 1986.

[Buh91] Peter Buhmann. *Disunifikation in modularen Termalgebren*. Master's thesis, Universität des Saarlandes, 1991. In preparation.

[CK90] C. C. Chang and H. J. Keisler. *Model Theory. Studies in Logic and the Foundations of Mathematics, vol. 73*, North-Holland Publishing Company, third edition, 1990.

[CL89] Hubert Comon and Pierre Lescanne. Equational problems and disunification. *Journal of Symbolic Computation*, 7(3,4):371–425, 1989.

[Der87] Nachum Dershowitz. Termination of rewriting. *Journal of Symbolic Computation*, 3:69–116, 1987.

[DoD81] United States Departement of Defense. *The Programming Language Ada. LNCS vol. 106*, Springer, 1981.

[EM85] H. Ehrig and B. Mahr. *Fundamentals of Algebraic Specification, vol. 1. EATCS-Monographs on Theoretical Computer Science*, Springer-Verlag, 1985.

[EM90] H. Ehrig and B. Mahr. *Fundamentals of Algebraic Specification, vol. 2. EATCS-Monographs on Theoretical Computer Science*, Springer-Verlag, 1990.

[End72] Herbert B. Enderton. *Mathematical Introduction to Logic.* Academic Press, 1972.

[Gre75] Sheila A. Greibach. *Theory of Program Structures: Schemes, Semantics, Verification. Lecture Notes in Computer Science, Vol. 35*, Springer Verlag, 1975.

[Gue79] Irène Guessarian. *Algebraic Semantics. Lecture Notes in Computer Science, Vol. 99*, Springer Verlag, 1979.

[HMM86] Robert Harper, David MacQueen, and Robin Milner. *Standard ML.* Technical Report ECS-LFCS-86-2, Edinburgh University, 1986.

[Kei71] H. Jerome Keisler. *Model Theory for Infinitary Logic. Studies in Logic and the Foundations of Mathematics, vol. 62*, North-Holland Publishing Company, 1971.

[KT90] D. Kozen and J. Tiuryn. Logics of programs. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science, volume B*, chapter 14, pages 789–840, Elsevier Science Publishers, 1990.

[LAB*81] Barbara Liskov, Rusell Atkinson, Toby Bloom, Eliot Moss, J. Craig Schaffert, Robert Scheifler, and Alan Snyder. *CLU reference manual. LNCS vol. 114*, Springer, 1981.

[LG86] Barbara Liskov and John Guttag. *Abstraction and Specification in Program Development.* MIT press, 1986.

[Lin69] P. Lindström. On extension of elementary logic. *Theoria*, 35:1–11, 1969.

[LL87] Thomas Lehmann and Jacques Loeckx. The specification language of OBSCURE. In D. Sannella and A. Tarlecki, editors, *5th Workshop on Specification of Abstract Data Types*, pages 131–153, Springer LNCS, vol. 332, 1987.

[LL90] Thomas Lehmann and Jacques Loeckx. *OBSCURE, A Specification Language for Abstract Data Types.* Technical Report A 19-90, Universität des Saarlandes, 1990. Submitted for publication.

[Loe87] Jacques Loeckx. Algorithmic specifications: a constructive specification method for abstract data types. *ACM Transactions on Programming Languages and Systems*, 9(4), 1987.

[LPP70] D. C. Luckham, D. M. R. Park, and M. S. Paterson. On formalized computer programs. *Journal of Computer and System Sciences*, 4:220–249, 1970.

[LS87] Jacques Loeckx and Kurt Sieber. *The Foundations of Program Verification*. Wiley/Teubner, 2nd edition, 1987.

[Mac86] David MacQueeen. Modules for standard ML. In [HMM86], 1986.

[Mah88] Michael J. Maher. Complete axiomatisations of the algebra of finite, rational and infinite trees. In *Third Anual Symposium on Logic in Computer Science*, pages 348–357, IEEE, Edinburgh, Scotland, july 1988.

[Mon76] J. Donald Monk. *Mathematical Logic. Graduate Texts in Mathematics, vol. 37*, Springer, 1976.

[Ros63] A. Rosenberg. On multi-head finite automata. In *Proceedings of the Fifth Annual Symposium on Switching Circuit Theory and Logical Design*, pages 221–228, 1963.

[Sha81] Mary Shaw, editor. *Alphard: Form and Content*. Springer, 1981.

[Ste90] Guy Steele. *Common LISP: The Language*. Digital Press, second edition, 1990.

[Tiu81] J. Tiuryn. A survey of the logic of effective definitions. In E. Engeler, editor, *Proceedings of the Workshop on Logics of Programs*, pages 198–245, Springer LNCS, vol. 125, 1981.

[Tre91] Ralf Treinen. *First order logic applied to first order data types*. PhD thesis, Universität des Saarlandes, 1991. In preparation.

[Tur85] David A. Turner. Miranda: a non-strict functional language with polymorphic types. In Jean-Pierre Jouannaud, editor, *IFIP International Conference on Functional Programming Languages and Computer Architecture*, pages 1–16, Springer LNCS, vol. 201, 1985.

[Tur86] David A. Turner. An overview of Miranda. *SIGPLAN notices*, 21(12):158–166, 1986.

[Vui74] Jean Vuillemin. Correct and optimal implementations of recursion in a simple programming language. *Journal of Computer and System Sciences*, 9:332–354, 1974.

[Wir85] Niklaus Wirth. *Programming in MODULA-2*. Springer, third edition, 1985.

[WLS76] W. A. Wulf, R. L. London, and M. Shaw. An introduction to the construction and verification of Alphard programs. *IEEE Transactions on Software Engineering*, 2(4):253–263, 1976.

[WPP*83] Martin Wirsing, Peter Pepper, Helmut Partsch, Walter Dosch, and Manfred Broy. On hierarchies of abstract data types. *Acta Informatica*, 20:1–33, 1983.