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.
Contents

1 Introduction 2

2 Preliminaries 4

3 Modules 5
   3.1 Syntax .......................... 5
   3.2 Semantics .......................... 8
   3.3 Basic Properties of the Semantics .......................... 11

4 Logic 13
   4.1 Basic Definitions and Properties .......................... 13
   4.2 Classes of Parameter Algebras .......................... 14
   4.3 Parameter Conditions .......................... 16

5 Properties of the New Logic 18
   5.1 Downward Skolem-Löwenheim .......................... 18
   5.2 Compactness .......................... 19

6 Decidability Questions 23
1 Introduction

The use of modules for data abstraction is now a well-established principle in software design, see for instance [Bis86]. From the programmer's 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 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 parameter 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\(^2\) ([Tur80], [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

\(^1\)Ada is a registered trademark of the United States Department of Defense.
\(^2\)Miranda is a registered trademark of Research Software Ltd.
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 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 unstructured 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 $\text{arity}(f) \in S^* \times S$. For $\text{arity}(f) = (s_1, \ldots, s_n, s)$ 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 $A \Gamma_{X, \Sigma}$ or shortly $\Gamma_{\Sigma}$ 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 $T_{\Sigma}$. 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 $\Sigma$-algebra $A$ consists of

- a set $s^A$ for each $s \in S$, called the carrier set of sort $s$, and
- a function $f^A : s^A_1, \ldots, s^A_n \to s^A$ for each function symbol $f \in F$ with $f : s_1, \ldots, s_n \to s$.

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

\[\quad\]
• for all $s \in S : s^B \subseteq s^A$
• for all $f \in F, f : s_1, \ldots, s_n \rightarrow s : f^B = f^A \upharpoonright s_1^B \times \ldots \times s_n^B$

If $\Sigma_1$ is a subsignature of $\Sigma_2$ and $A$ a $\Sigma_2$-algebra then $A \upharpoonright \Sigma_1$ denotes the restriction of $A$ to $\Sigma_1$. A $\Sigma$-algebra $A$ induces an interpretation function for terms:

$$A : T_\Sigma(X) \rightarrow (\Gamma_{X, A} \rightarrow A)$$

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 $C$ of algebras we write $C \models w$ if $A \models w$ for each $A \in C$.

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 assigns $a$ to $c$.

$\equiv$ 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 $\Sigma$-algebra $A$ is an elementary submodel of a $\Sigma$-algebra $B$ if $A \subseteq B$ and for all $\Sigma$-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 being 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 \(\text{bool}\) and
- \(F\) contains the following function symbols:

\[
\begin{align*}
\text{true} & : \rightarrow \text{bool} \\
\text{false} & : \rightarrow \text{bool} \\
\bot_s & : \rightarrow s \quad \text{for all } s \in S \\
\text{ifthenelse}_s & : \text{bool}, s, s \rightarrow s \quad \text{for all } s \in S \\
=_{s} & : s, s \rightarrow \text{bool} \quad \text{for all } s \in S
\end{align*}
\]

In [Loe87] \(\bot_s\) and \(\text{ifthenelse}_s\) 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 implicitly define functions \(\bot_s\), \(=_{s}\) and \(\text{ifthenelse}_s\) 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 \(\text{is}_c\) and selector functions \(\text{select}_c\). This bears some similarities with the list data structure of Common LISP ([Ste90]) where \(\text{nil}\) and \(\text{cons}\) are constructors, \(\text{consp}\) is a test predicate and \(\text{car}\) and \(\text{cons}\) 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.

\(^3\)This is of course no complete analogy since LISP does not obey a strong typing discipline.
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 \(\Sigma_{P_m}\). \(PS_m\) 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:

\[
\begin{align*}
\{ & \text{ifthenelse}_s : \text{bool}, s, s \to s \mid s \in NS_m \\
\cup & \{ =_s : s, s \to \text{bool} \mid s \in NS_m \}
\cup & \{ \text{is}_c ? : s \to \text{bool} \mid s \in NS_m \}
\cup & \{ \text{select}_i^c : s \to s_j \mid c \in K_m, c : s_1, \ldots, s_j, \ldots, s_n \to s \}
\cup & \{ \bot_s : \to s \mid s \in NS_m \}
\end{align*}
\]

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

- \(PR_m\) is a recursive program of the form

\[
\begin{pmatrix}
  f_1(x_{1,1}, \ldots, x_{1,t_1}) & \iff & t_1 \\
  \vdots & \vdots & \vdots \\
  f_n(x_{n,1}, \ldots, x_{n,t_n}) & \iff & t_n
\end{pmatrix}
\]

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

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

The pair \((\Sigma_{P_m}, \Sigma_{E_m})\) 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 assigns the intended meanings to the standard parts of the signature.

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

- $\text{bool}^A = \{\text{true}, \text{false}, \bot_{\text{bool}}\}$ and $\text{true}^A = \text{true}, \text{false}^A = \text{false}, \bot_{\text{bool}}^A = \bot_{\text{bool}}$

- for all $s \in S$ and $x_1, x_2 \in s^A$:
  \[\text{ifthenelse}^A_s(\text{true}, x_1, x_2) = x_1\]
  \[\text{ifthenelse}^A_s(\text{false}, x_1, x_2) = x_2\]
  \[\text{ifthenelse}^A_s(\bot_s, x_1, x_2) = \bot_s^A\]

- For each function symbol $f \in F$ the denotation $f^A$ is continuous with respect to the following ordering $\sqsubseteq_s$:
  \[x_1 \sqsubseteq_s x_2 \text{ iff } x_1 = x_2 \text{ or } x_1 = \bot_s^A\]

Alg$_\Sigma$ denotes the class of all standard algebras with signature $\Sigma$. 
Note that, since a standard signature contains a constant symbol $\bot_s$ of each sort $s$, a standard algebra always contains a distinguished carrier $\bot_s^A$ 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 $(\Sigma_P, \Sigma_E)$ and a $\Sigma_P$-standard algebra $A$ to a $\Sigma_E$-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 $(\Sigma_P, \Sigma_E)$ and $A \in \text{Alg}_{\Sigma_P}$. We define a family of sets $(D_{s, m}^A)_{s \in AS_m}$ as follows:

- For each $s \in PS_m$ let $D_{s, m}^A := S^A - \{\bot_s^A\}$
- For all $s \in NS_m$ define sets $D_{s, m}^A$ by simultaneous induction:
  - if $k \in K_m$, $k : s_1, \ldots, s_n \rightarrow s$ and $d_i \in D_{s_i, m}^A$ for $i = 1, \ldots, n$
  - then $k(d_1, \ldots, d_n) \in D_{s, m}^A$
- Now define for all $s \in AS_m$: $D_{s, m}^A := D_{s, m}^A \cup \{\bot_s\}$

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 $(\Sigma_P, \Sigma_E)$ and $A \in \text{Alg}_{\Sigma_P}$. We define an algebra $A^* \mid_{\Sigma_P}$ with signature $(AS_m, AF_m \setminus NF_m)$ as follows:

- $A^* \mid_{\Sigma_P} := A$
- $s^{A^*} := D_{s, m}^A$ for all $s \in NS_m$
- $\bot_s^{A^*} := \bot_s$ for all $s \in NS_m$
- For all $s \in NS_m$ if then else, and $=$, obtain their meaning according to the definition of standard algebra.
- For all $k : s_1, \ldots, s_n \rightarrow s \in K_m$:
  $$k^{A^*}(x_1, \ldots, x_n) := \begin{cases} k(x_1, \ldots, x_n) & \text{if } x_i \neq \bot_s^A \text{ for all } i \\ \bot_s & \text{otherwise} \end{cases}$$
- For all $c : s_1, \ldots, s_n \rightarrow s \in K_m$:
  $$\text{is}_c^{A^*}(x) := \begin{cases} \text{true} & \text{if } x = c(x_1, \ldots, x_n) \text{ for some } x_i \\ \bot_{\text{bool}} & \text{if } x = \bot_s^A \\ \text{false} & \text{otherwise} \end{cases}$$
• For all \(c: s_1, \ldots, s_n \rightarrow s \in K_m\):

\[
\operatorname{select}_c^{A^*}(x) := \begin{cases} 
  x_j & \text{if } x = c(x_1, \ldots, x_j, \ldots, x_n) \\
  \bot_{s_i}^A & \text{for some } x_i \neq \bot_{s_i}, i = 1 \ldots n \\
  \bot_{s_j} & \text{otherwise}
\end{cases}
\]

Lemma 1: \(A^*\) is a standard algebra.

Proof: This follows from the continuity of the sequential \(\text{if-then-else}\) function ([LS87]) and the strictness of the remaining functions. \(\square\)

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 \(\bot\), this yields a term that is assigned a meaning by the algebra \(A^*\).

Definition 7: Let \(m\) be a module with signature \((\Sigma_P, \Sigma_E)\) and \(t \in T_{\Sigma_{m}}(X)\). Then for each \(n \in \mathbb{N}\), \(t^n(n)\) 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 \(\bot\).

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 taken). 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 \((\Sigma_P, \Sigma_E)\). Then \(\mathcal{M}(m)\) is a function

\[
\mathcal{M}(m): \text{Alg}_{\Sigma_P} \rightarrow \text{Alg}_{\Sigma_E}
\]

where for all \(A \in \text{Alg}_{\Sigma_P}:

• \(\mathcal{M}(m)(A) \mid (\text{AS}_m, \text{EF}_m \setminus \text{NF}_m) := A^* \mid (\text{AS}_m, \text{EF}_m \setminus \text{NF}_m)\)

• for all \(f: s_1, \ldots, s_n \rightarrow s \in \text{NF}_m \cap \text{EF}_m, a_i \in s_i^{A^*}:

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

for an arbitrary assignment \(\alpha \in \Gamma_{A^*}\).
The choice of the assignment $\alpha$ 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]. \hfill \Box

**Lemma 3** Let $m$ be a module with signature $(\Sigma_P, \Sigma_E)$, $A \in \text{Alg}_{\Sigma_P}$, $B = \mathcal{M}(m)(A)$, $f : s_1, \ldots, s_n \to s \in EF \cap NF_m$ 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 $f^B(a_1, \ldots, a_n) = \bot_1^B$ and $B(f(x_1, \ldots, x_n)(j))(\alpha) = \bot_1^B$ for all $j \in \mathbb{N}$
- or $f^B(a_1, \ldots, a_n) = c \neq \bot_1^B$ and there is a $j_0$ such that for all $j \geq j_0$: $B(f(x_1, \ldots, x_n)(j))(\alpha) = c$

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

### 3.3 Basic Properties of the Semantics

Our semantics obeys the persistence 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 $(\Sigma_P, \Sigma_E)$ and $A \in \text{Alg}_{\Sigma_P}$. 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 $(\Sigma_P, \Sigma_E)$, $A, B \in \text{Alg}_{\Sigma_P}$ 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. \hfill \Box
Lemma 6 Let m be a module with signature \((\Sigma_P, \Sigma_E), \Sigma_P \subseteq \Sigma_E, A \in \text{Alg}_{\Sigma_P}\) and \(B' \in \text{Alg}_{\Sigma_E}\) 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 \(s \in \text{NS}\):

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 \(s \in \text{NS}\):

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, \(x_1, \ldots, x_n\) 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 \(B' = B''\). \(\Box\)
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 \( \text{WFF}_\Sigma \) be the set of first order sentences over the language \( (S, F, \{=_s \mid s \in S\}) \) where each \( =_s \) is a binary predicate symbol.

If \( m \) is a module with signature \((\Sigma_P, \Sigma_E)\) we denote \( \text{WFF}_{\Sigma_P} \) by \( \text{PWFF}_m \) and \( \text{WFF}_{\Sigma_E} \) by \( \text{WFF}_m \).

The symbol \( =_s \) is overloaded here: we use it as a function symbol with arity \( s, s \to \text{bool} \) 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:

\[
(t_1 \subseteq t_2) \quad \text{stands for} \quad (t_1 = \bot \lor t_1 = t_2)
\]

\[
\forall x \in s \cdot w \quad \text{stands for} \quad \forall x : s \cdot x = \bot \lor w
\]

\[
\exists x \in s \cdot w \quad \text{stands for} \quad \exists x : s \cdot x \neq \bot \land w
\]

We now come to the central definition of this paper. For a sentence \( w \in \text{WFF}_\Sigma \) 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 semantics of a module with signature \((\Sigma_P, \Sigma_E)\) in order to express properties of standard \( \Sigma_P \)-algebras by \( \Sigma_E \)-sentences.

Definition 10 Let \( m \) be a module with signature \((\Sigma_P, \Sigma_E)\), \( A \in \text{Alg}_{\Sigma_P} \) and \( w \in \text{WFF}_m \). We define

\[
A \models_m w \quad \Leftrightarrow \quad \mathcal{M}(m)(A) \models w
\]

For \( W \subseteq \text{WFF}_m \) 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 \( A \in \text{Alg}_{\Sigma_P} \). Furthermore \( \text{Th}_m(A) := \{ w \in \text{WFF}_m \mid A \models_m w \} \)

For example let \( m \) be the module of Figure 1 and \( \text{Nat} \) the extension of the algebra of natural numbers to a standard algebra. Then

\[
\text{Nat} \models_m \forall l_1, l_2 : \text{list} \cdot \text{sum} (\text{app}(l_1, l_2)) = \text{sum}(l_1) + \text{sum}(l_2)
\]
As an immediate consequence of the persistency of the semantics (Lemma 4) we get

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

$$A \models w \iff A \models_m w$$

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 $\text{Alg}_{\Sigma_P}$. Instead it is often natural to restrict the parameter algebras to some subclass of $\text{Alg}_{\Sigma_P}$. 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 $C$ of $\Sigma$-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 $C$
- then $W$ has a model in $C$

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

**Definition 12** A domain operator $\mathcal{D}$ maps each standard signature $\Sigma$ to a subclass $\mathcal{D}_\Sigma$ of $\text{Alg}_{\Sigma}$ such that the following holds:

1. $\mathcal{D}_\Sigma$ is compact.
2. $\mathcal{D}_\Sigma$ is closed under elementary submodels.
3. For any sort symbol $s$ in $\Sigma$ and constant symbol $c$ not in $\Sigma$:

$$\mathcal{D}_\Sigma \cup \{c.a\} = \{(A, a) \mid A \in \mathcal{D}_\Sigma \text{ and } a \in s^A\}$$

The following mappings are no domain operators:
\( \forall x: \text{bool} . \ (x = \bot_{\text{bool}} \lor x = \text{true} \lor x = \text{false}) \)

(2) \( \bot_{\text{bool}} \neq \text{true} \)

(3) \( \bot_{\text{bool}} \neq \text{false} \)

(4) \( \text{true} \neq \text{false} \)

(5) \( \forall x, y : s . \ [(x =_s y) = \text{true} \ \forall s . (x = y \land x \neq \bot_s)] \)

(6) \( \forall x, y : s . \ [(x =_s y) = \text{false} \ \forall s . (x \neq y \land x \neq \bot_s \land y \neq \bot_s)] \)

(7) \( \forall x, y : s . \ [(x =_s y) = \bot_{\text{bool}} \ \forall s . (x = \bot_s \lor y = \bot_s)] \)

(8) \( \forall x, y : s . \text{if } \text{true} \text{ then } x \text{ else } y \ fi = x \)

(9) \( \forall x, y : s . \text{if } \text{false} \text{ then } x \text{ else } y \ fi = y \)

(10) \( \forall x, y : s . \text{if } \bot_{\text{bool}} \text{ then } x \text{ else } y \ fi = \bot_s \)

(11) \( \forall \vec{x}, \vec{y} : s . \ [(\land_{i=1\ldots n} (x_i \subseteq y_i) \supset f(x_1, \ldots, x_n) \subseteq f(y_1, \ldots, y_n)] \)

Figure 3: Axiom schemes for standard algebras.

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 \( \Sigma \) \( F_\Sigma \) does not fulfill compactness.

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 \( \Sigma \) 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 \( \iota \) 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 \( \{\text{true}, \text{false}\} \cup \{\bot_s \mid s \in S\} \) are allowed. Then the operator mapping each signature \( \Sigma \) 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 exactly 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. \qed

As a consequence the following operators are indeed domain operators:

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

2. The operator $\exists^{strict}$ 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 explicitly 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 $M(m)(A) \in \exists_{\Sigma_k}$ for $A \in \exists_{\Sigma_p}$.

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 $(\Sigma_P, \Sigma_E)$, $\exists$ a class operator and $w \in WFF_m$. A sentence $v \in PWFF_m$ is a $\exists, m$-parameter condition of $w$ if for all $A \in \exists_{\Sigma_p}$:

$$A \models v \implies A \models_m w$$

A sentence $v \in PWFF_m$ is a $\exists, m$-weakest parameter condition of $w$ if for all $A \in \exists_{\Sigma_p}$:

$$A \models v \iff A \models_m w$$

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 $(\Sigma_P, \Sigma_E)$, $\exists$ a domain operator and $w \in WFF_m$. Then for all weakest parameter conditions $v_1, v_2 \in PWFF_m$: $\exists_{\Sigma_P} \models v_1 \equiv v_2$
PAR  SORTS elem
OPNS 0 → elem
  pred: elem → elem
BODY  FCTS  issstandard: elem → bool
  PROG  issstandard(x) ← if x = 0 then true
       else issstandard(pred(x))

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

PAR  SORTS elem
BODY  SORTS list
  CONS  nil: → list
        cons: elem, list → list
  FCTS  isin: elem, list → bool
  PROG  isin(e, l) ← if is-nil?(l) then false
       else if select$^1_{cons}(l) = e$ then true
       else isin(e, select$^2_{cons}(l))

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

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

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

   $(\forall x \in elem.0 + x = x \wedge x + 0 = x) \land
   (\forall l_1, l_2 \in list.\text{sum(app}(l_1, l_2)) = \text{sum}(l_1) + \text{sum}(l_2))$

has the $\mathcal{Q}^{\text{strict}}$, m-weakest parameter condition:

   $(\forall x \in elem.0 + x = x \wedge x + 0 = x) \land
   (\forall x_1, x_2, x_3 \in elem.x_1 + (x_2 + x_3) = (x_1 + x_2) + x_3)$

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

   $w := \forall x \in elem.\text{issstandard}(x) = \text{true}$

does not have a $\mathcal{Q}^{\text{strict}}$, 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.

17
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

\[
    w := \exists l \in \text{list} . \forall x \in \text{elem}. \text{isin}(x, l) = \text{true}
\]

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

**Lemma 10** Let \( m \) be a module with \( NF_m = \emptyset \) and \( \exists \) a domain operator. Then for each \( w \in WFF_m \) there exists a \( \exists, m \)-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 [Tec91] and [Buh91] for details how to apply these results to the semantics of modules. \( \square \)

## 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-Löwenheim 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 $(\Sigma_P, \Sigma_E)$, $\mathcal{Z}$ a domain operator and $A \in \mathcal{Z}_{\Sigma_P}$.

Then for each family $Z = (Z_s)_{s \in P_S}$ of sets with cardinality at most $\aleph_0$ and $Z_s \subseteq s^A$ there is a $B \in \mathcal{Z}_{\Sigma_P}$ 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 $\mathcal{Z}_{\Sigma_P}$ under elementary submodels $B \in \mathcal{Z}_{\Sigma_P}$. $\square$

**Corollary 1** Let $m$ be a module with signature $(\Sigma_P, \Sigma_E)$ and $A \in \mathcal{Z}_{\Sigma_P}$ of infinite cardinality. Then there is a $B \in \mathcal{Z}_{\Sigma_P}$ 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 infinite 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 $(\Sigma_P, \Sigma_E)$ and $\mathcal{Z}$ a domain operator. We say that $\mathcal{Z}_{\Sigma_P}$ is $\mathcal{Z}$-compact if for each $W \subseteq WFF_m$ the following
holds:

If for each finite \( F \subseteq W \) there is a \( A \in \mathfrak{S}_{\Sigma_p} \) with \( A \models_m F \), then there exists \( B \in \mathfrak{S}_{\Sigma_p} \) with \( B \models_m W \)

**Theorem 2** Let \( m \) be a module and \( \mathfrak{S} \) a domain operator. Then the following statements are equivalent:

1. For each extension \( m' \) of \( m \), \( m' \models \mathfrak{S} \)-compact.

2. For each extension \( m' \) of \( m \) and \( w \in WFF_{m'} \), \( w \) has a \( \mathfrak{S}, m' \)-weakest parameter condition

**Proof:**

\((1) \Longleftarrow (2)\)

This is an easy consequence of the definition of a weakest parameter condition and of the compactness property of the domain operator \( \mathfrak{S} \).

\((1) \implies (2)\)

Assume that for each extension \( m' \) of \( m \), \( m' \models \mathfrak{S} \) 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 \( \subseteq \) on \( W \) by

\[
w_1 \subseteq w_2 : \iff (\phi_1(w_1), \phi_2(w_1)) \leq_{\text{lex}} (\phi_1(w_2), \phi_2(w_2))
\]

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 \( \subseteq \) is a well founded quasi ordering ([Der87]).
Now let \( w \in W \) be a minimal formula with respect to \( \subseteq \) such that there exists an extension \( m' \) of \( m \) with \( w \in WFF_{m'} \) and \( w \) does not have a weakest \( \exists, m' \)-parameter condition. \( \Sigma' \) 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 \( \exists, m^* \)-weakest parameter condition where \( m^* \) is the extension of \( m \) by the constants occurring in \( v \).

1. Suppose \( w = \exists x : s \cdot v \) 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 \( \exists y : s \cdot r(c/y) \) 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 \( A \in \Theta_{\Sigma'} \).

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

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

\[
C_s := T_{K,s}(X_{\text{par}}) \cup \{ \bot_s \}
\]

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

\[
M(m')(A)(t)(\alpha) = a
\]

(1)

For each finite set \( F \subseteq C_s \), the formula

\[
\bigvee_{t \in F} \exists \text{free}(t) \cdot v(x/t)
\]

has by minimality of \( w \) a \( \exists, m' \)-weakest parameter condition \( r_F \) which is itself a \( \exists, m' \)-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 \( A \in \Theta_{\Sigma'} \) with

\[
A \models_{m'} \{ \exists x : s \cdot v \} \cup \{ \forall \text{free}(t) \cdot \neg v(x/t) \mid t \in F \}
\]

21
Since $\models m$, is compact there is an $A \in \mathcal{S}_t$ with

$$A \models m'\{\exists x : s \cdot v\} \cup \{\forall \text{free}(t) \cdot \neg v(x/t) \mid t \in C_s\}$$

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 $w = v_1 \lor v_2$. By minimality of $w$ $v_1$ and $v_2$ have weakest parameter conditions $r_1$ and $r_2$, respectively. Then $r_1 \lor r_2$ 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

$$\left(\underbrace{t_1 = t_2}_v \land \underbrace{t_1 \neq \bot}_v \right) \lor \left(\underbrace{\neg t_1 \neq \bot}_v \lor \underbrace{t_2 \neq \bot}_v\right)$$

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 $\mathcal{S}, m'$-weakest parameter condition $r_n$ of $t_1(n) = t_2(n) \land t_1(n) \neq \bot$. Observe that $\models m' \neg r_n \lor \neg r_m$ 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 $A \in \mathcal{S}_t$ with

$$A \models m'\{v_1\} \cup \{-r_n \mid n \in F\}$$

By the compactness property of $\models m'$, there is a $A \in \mathcal{S}_t$ with

$$A \models m'\{t_1 = t_2 \land t_1 \neq \bot\} \cup \{t_1(n) \neq t_2(n) \lor t_1(n) = \bot \mid n > 0\}$$

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

- For all $n$: $M(m')(A)(t_1(n)) = \bot$. This contradicts $M(m')(A)(t_1) \neq \bot$ by Lemma 3.

- There is a $n_0$ such that $M(m')(A)(t_1(n_0)) \neq \bot$. Then for all $n > n_0$ $M(m')(A)(t_1(n)) \neq \bot$ and therefore $M(m')(t_1(n)) \neq M(m')(t_2(n))$. This contradicts $M(m')(A)(t_1) = M(m')(A)(t_2)$ 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 $\Sigma_n$ that carries each signature to the class of finite standard algebras with cardinality less or equal to the fixed number $n$. In any $\Sigma_n \Sigma$ 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 $\Sigma_n$ 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 $\Sigma_m \mu$-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 $\Sigma$ 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 $\Sigma$-equations
- $T$ is hierarchy-persistent

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

The extension of A to a standard algebra is obtained by extending the signature to a standard signature, assigning $\bot$, 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 \(\delta\) is a transition function

\[\delta: (Q_1 \cup Q_2 \cup \{q_0\}) \times \{0, 1\} \rightarrow Q_1 \cup Q_2 \cup \{q_a, q_r\}\]

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. \(\square\)

The module of Figure 6 simulates a THA in the following sense. To a given \(\Sigma_P\) algebra \(B\) we associate the input tape \(\text{tape}_B\) that is defined by

\[\text{tape}_B(i) := \begin{cases} 0 & \text{if } B(\text{contents}_0(\text{next}^i(\text{start}))) = \text{true} \\ 1 & \text{if } B(\text{contents}_0(\text{next}^i(\text{start}))) = \text{false} \end{cases}\]

Obviously each possible input tape is a \(\text{tape}_B\) for some \(\Sigma_P\) algebra \(B\).
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 $B \in \text{Alg}_{\Sigma_p}$ and $\alpha \in \Gamma_B$:

$tape_B \in L_A \Rightarrow M(m)(B)(F_{q_0}(\text{start}, \text{start}, x))(\alpha) = M(m)(B)(\text{if } test(x) \text{ then } true \text{ else } F_{q_0}(\text{start}, \text{start}, f(x)))(\alpha)$

$tape_B \notin L_A \Rightarrow M(m)(B)(F_{q_0}(\text{start}, \text{start}, x))(\alpha) = \bot$

**Proof:** This follows easily from the definitions. \qed
Lemma 18 Let $m$ be the module associated to the THA $A$ according to Figure 6 and $\mathfrak{S}$ a non trivial domain operator.

1. If $L_A = \emptyset$ then $\models_m H = \bot$

2. If $L_A \neq \emptyset$ then the formula $(H \neq \bot)$ does not have a $\mathfrak{S},m$-weakest parameter condition.

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

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

$$e_t := \bigwedge_{i=0 \ldots n} \left\{ \begin{array}{ll}
\text{contents}0(\text{next}^i(\text{start})) = \text{true} & \text{if } t(i) = 0 \\
\text{contents}0(\text{next}^i(\text{start})) = \text{false} & \text{if } t(i) = 1
\end{array} \right.$$

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

$$B \models_m H \neq \bot \iff B \models \text{test}(f^i(a)) = \text{true} \text{ for some } i \quad (2)$$

On the other hand each set of the form

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

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

$$E = \{e_t\} \cup \{\text{test}(f^i(a)) = \text{false} \mid i \leq n_0\} \cup \{\text{test}(f^{n_0+1}(a)) = \text{true}\}$$

By compactness of $\mathfrak{S}$ there is an algebra in $\mathfrak{S}_{\Sigma_P}$ satisfying

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

This contradicts (2). \qed

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

Theorem 3 For a non-trivial domain operator $\mathfrak{S}$ the following sets are not semidecidable:
PAR  SORTS tapeposition  
         value  
OPNS start: → tapeposition  
         next: tapeposition → tapeposition  
         contents0: tapeposition → bool  
         a: → value  
         f: value → value  
         test: value → bool  

BODY  FCTS  H: → bool  
         F_q: tapeposition, tapeposition, value → bool  
         for all states q of the automaton  

PROG  H  
         ⇐ F_q0(start, start, a)  
         F_q(p_1, p_2, x)  
         ⇐ if test(x) then true  
         else if contents0(p_1)  
         then F_{q,0}(next(p_1), p_2, f(x))  
         else F_{q,1}(next(p_1), p_2, f(x))  
         for all states q ∈ Q_1  
         F_q(p_1, p_2, x)  
         ⇐ if test(x) then true  
         else if contents0(p_2)  
         then F_{q,0}(p_1, next(p_2), f(x))  
         else F_{q,1}(p_1, next(p_2), f(x))  
         for all states q ∈ Q_2  

         F_{q0}(p_1, p_2, x)  
         ⇐ true  
         F_a(p_1, p_2, x)  
         ⇐ true  

EXPORT  H: → bool  

---  

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

- the set of modules m such that all formulas w ∈ WFF_m have a ∃, m-weakest parameter condition
- the set of pairs (w, m) where m is a module, w ∈ WFF_m and w has a ∃, 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 $\mathcal{S}$ a non trivial domain operator.

1. If $D_A = \emptyset$ then $\models_m H = \text{true}$

2. If $D_A \neq \emptyset$ then the formula $(H \neq \bot)$ does not have a $\mathcal{S}, m$ weakest parameter condition.

**Theorem 4** For a non-trivial domain operator $\mathcal{S}$ the following sets are not semidecidable:

- the set of modules $m$ such that some formula $w \in \text{WFF}_m$ does not have a $\mathcal{S}, m$ weakest parameter condition

- the set of pairs $(w, m)$ where $m$ is a module and $w \in \text{WFF}_m$ and $w$ does not have a $\mathcal{S}, m$ weakest parameter condition

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

**References**


29


