Abstract

Abstract Data Types (ADTs) can be specified by the Classified Model (CM) specification language—a first-order Horn language with equality and sort “classification” assertions. Sort assertions generalize the traditional syntactic signatures of ADT specifications, resulting in all of the specification capability of traditional equational specifications, but with the improved expressibility of the Horn-with-equality language.

This work extends corresponding results from Many-Sorted Algebra (MSA), Order-Sorted Algebra (OSA) and Order-Sorted Model (OSM) specification techniques by promoting their syntactic signatures to assertions in the Classified-Model specification language, yet retaining sorted quantification. It is shown how this solves MSA problems such as error values, polymorphism and subtypes in a way different from the OSA and OSM solutions. However, the CM technique retains the MSA and order-sorted approach to parameterization. CM proof theory and semantics are developed, including theorems for soundness, completeness and the existence of a free model.

1 Introduction

A classified model is the intended meaning of an abstract data type (ADT) specification written in a specification language that defines sorts by assertions rather than by the syntactic signatures of conventional ADT specification techniques. A specification in the proposed language consists of a set of first-order Horn formulas (including the equality predicate) that are universally quantified over the defined sorts.

For example, the universally quantified atomic assertions and equations in Figure 1 specify the natural numbers with the operations successor, addition and multiplication. The first two assertions (declaration assertions) inductively define the elements of sort \( \text{nat} \) by asserting in the base case that 0 is an element of \( \text{nat} \) and in the inductive step that the successor of a natural number is also a natural number. Universally quantified variables are subscripted with a sort identifier to indicate that the variable may range over all objects that are asserted by the specification to be of that sort. The remaining assertions (equations) define the binary operations using the same base and inductive cases by identifying (equating) elements of \( \text{nat} \) according to the usual meaning of these operations.

The software structure \( \text{STACK-OF-NAT} \) is specified in Figure 2 by extending the specification \( \text{NAT} \) with additional assertions. The extension introduces \( \text{stk} \) as another sort of fundamental object and defines it by base and inductive cases in the first two assertions. The remaining assertions define the operations \( \text{top} \) and \( \text{pop} \), which respectively return and remove the most recently \( \text{push}-\text{ed} \) \( \text{nat} \) object.

Classified-model (CM) specifications can be used like the familiar axiomatic theories of mathematics (e.g., groups and rings) to prove properties of the underlying objects specified or to compute with these objects. For example, using rules which include the usual algebraic rules of substitution of an expression for any variable and replacement of a term by an equal term, we can prove from the above \( \text{STACK-OF-NAT} \) specification the assertion

\[
(T_{\text{stk}}) \ \text{nat}(\text{if } T_{\text{stk}} = \text{stknil} \ \text{then } 0 \ \text{else } \text{top}(t_{\text{stk}}))
\]
$$(X_{\text{nat}}) \quad \text{nat}(0)$$

$$(X_{\text{nat}}) \quad \text{nat}(s(X_{\text{nat}}))$$

$$(X_{\text{nat}}) \quad X_{\text{nat}} + 0 = X_{\text{nat}}$$

$$(X_{\text{nat}}, Y_{\text{nat}}) \quad X_{\text{nat}} + s(Y_{\text{nat}}) = s(X_{\text{nat}} + Y_{\text{nat}})$$

$$(X_{\text{nat}}) \quad X_{\text{nat}} \times 0 = 0$$

$$(X_{\text{nat}}, Y_{\text{nat}}) \quad X_{\text{nat}} \times s(Y_{\text{nat}}) = (X_{\text{nat}} \times Y_{\text{nat}}) + X_{\text{nat}}$$

Figure 1: Classified Model specification (NAT) for naturals (nat).

$$(S_{\text{stk}}, X_{\text{nat}}) \quad \text{stk}(\text{push}(S_{\text{stk}}, X_{\text{nat}}))$$

$$(S_{\text{stk}}, X_{\text{nat}}) \quad \text{top}(\text{push}(S_{\text{stk}}, X_{\text{nat}})) = X_{\text{nat}}$$

$$(S_{\text{stk}}, X_{\text{nat}}) \quad \text{pop}(\text{push}(S_{\text{stk}}, X_{\text{nat}})) = S_{\text{stk}}$$

Figure 2: Classified Model specification STACK-OF-NAT.

that the if-then-else is of sort nat, given the usual definition of the if-then-else function. When $T_{\text{stk}}$ is not the empty stack, stknil, then the sort of the if-then-else expression is the sort of top($T_{\text{stk}}$), i.e., sort nat according to the specification. The sort of top(stknil) is not defined by the stack specification, but the if-then-else has the (arbitrary) nat value 0 when $T_{\text{stk}}$ is stknil, so the if-then-else expression is also of sort nat in this case. Using an induction rule, to be described later, we can also prove from the specification NAT the assertion

$$(X_{\text{nat}}, Y_{\text{nat}}) \quad \text{nat}(X_{\text{nat}} + Y_{\text{nat}})$$

by proving

$$(X_{\text{nat}}) \quad \text{nat}(X_{\text{nat}} + 0)$$

and

$$(X_{\text{nat}}) \quad \text{nat}(X_{\text{nat}} + s(a))$$

from the specification, the declaration nat($a$) and the induction hypothesis

$$(X_{\text{nat}}) \quad \text{nat}(X_{\text{nat}} + a)$$

In a similar way, the induction rule can be used to prove other important NAT assertions such as the associative, commutative and distributive laws. The order of proofs is important: assertions proved by one application of induction are added to the specification and used in subsequent induction proofs.

2 The Problem: Data types are not the same as MSAs

The roots of the specification technique described in this work are in algebra and specifically in many-sorted, or heterogeneous, algebra [1, 6]. An algebra is a set along with operations over the set. Data types have traditionally been defined as a set of data domains, or carriers, which have named constants and operations over the carriers. Furthermore, the carriers are generated from the constants by use of the operators. A many-sorted algebra is a set of disjoint sets along with
operations over the sets. Data types have often been studied formally as many-sorted algebras, but data types are not just many-sorted algebras. A number of problems encountered in the interpretation of data types as many-sorted algebras cannot be resolved within the many-sorted algebra formalism. First, the “constructor-selector” \[4\] problem cannot be solved within the MSA formalism. Other problems involve

- error values,
- polymorphism and
- subsorts.

Error values result from nonsensical combinations of operations. Recall that in the MSA formalism the sort of an operation is dictated by its signature so that, for example, in the $STACK\text{-OF}\text{-}NAT$ MSA specification the top of any stack, including the empty stack, is a natural number. There are several ways of dealing with the problem. We can ignore the problem by assigning default values to the problem combinations, e.g., $top(stknil) = 0$, but then it is very hard to even formulate the concept of a safe program. Alternatively, we can introduce special error objects explicitly, but then the specification is complicated by the requirement to qualify some equations with preconditions or to distinguish “ok” equations from “error” equations.

Polymorphism (overloading) is the use of the same operation symbol with various argument and result sorts. Because in the MSA formalism the sort of each operation symbol is dictated by the signature, we must use a different operation symbol for each different argument and result sort. For example, the addition operator “+” is used in mathematics and in most programming languages as an overloaded operator for natural numbers, integers, rationals, etc., but in the MSA formalism we must use a different symbol for each different use. To make matters worse, each of these operations needs its own copy of the “generic” specifying equations that are given above.

Subsorts are not allowed in the MSA formalism. The resolution of this problem is fundamental to the solution of the two previous problems. If, for example, we consider the natural numbers to be a subsort of all the possible values returned by the $STACK\text{-OF}\text{-}NAT$ operation $top$, then it makes sense to consider $top(stknil)$ as just a member of the superset. Similarly, if we consider the natural numbers as a subset of the integers and the integers as a subset of the rationals, we can easily consider a polymorphic addition operator.

The main problem with the MSA formalism is that its sort structure is based on a simplistic approach to sorts in which each operator is sorted statically in a signature declaration. In the beginning of the ADT research effort many-sorted algebras may have appeared to be an appropriate formalism for data types, but over time enough troublesome cases have been presented to warrant the consideration of more general formalisms. The solution suggested in the classified-model approach is to generalize the notion of sort so that different terms constructed from the same operation symbol can have different sorts and so that a single term, or set of terms, can have more than one sort. This also induces a subsort relation among sorts.

3 Order-Sorted Algebra (OSA) and Order-Sorted Model (OSM)

Order-sorted algebra (OSA) \[2\] techniques generalize MSA by providing a subsort partial ordering among the sorts. Operator overloading and nonsensical application of function symbols, “errors”, are handled by restricting function to subsorts. Order-sorted model (OSM) techniques generalize OSA by allowing predicates and Horn formulas instead of just conditional equations.

For example, a stack-of-natural, with subsort $nstk$ standing for non-empty stack, is specified in the order-sorted specification language OBJ2 of Figure 3. The first intended statement indicates that the natural numbers $nat$ are neither extended nor contracted, i.e., an OBJ2 specification $NAT$ is enriched to $STACK\text{-OF}\text{-}NAT$. The next two statements indicate that $stk$ (a stack) and $nstk$ (non-empty stack) are sorts, where $nstk$ is a subset of $stk$. The lines starting with
“op” form a syntactic signature that specifies, as in MSA, all allowed combinations of function (operation) symbols and arguments. The “var” statements declare variables for the following equations indicating combinations which represent the same value. Sound and complete rules of inference exist for OSM [5] and include the MSA and OSA systems as special cases.

\[
\text{obj \hspace{1em} STACK-OF-NAT is} \\
\hspace{1em} \text{protecting nat.} \\
\hspace{1em} \text{sorts stk, nstk.} \\
\hspace{1em} \text{subsorts nstk < stk.} \\
\hspace{1em} \text{op stknil :\to stk.} \\
\hspace{1em} \text{op push : stk nat \to nstk.} \\
\hspace{1em} \text{op top : nstk \to nat.} \\
\hspace{1em} \text{op pop : nstk \to stk.} \\
\hspace{1em} \text{var I : nat.} \\
\hspace{1em} \text{var S : stk.} \\
\hspace{1em} \text{eq \hspace{1em} \text{top(push(S,I)) = I.}} \\
\hspace{1em} \text{eq \hspace{1em} \text{pop(push(S,I)) = S.}} \\
\]  

Figure 3: Order-Sorted OBJ2 STACK-OF-NAT specification.

The Order-Sorted methods are an improvement over the many-sorted techniques since some of the problems mentioned above are solved. But, the subsorts of OSA induce a new problem which does not arise with the old “pigeon-holing” MSA approach in which each term has exactly one sort. It is not possible to reason about the sort of the value of an expression, i.e., about special values an expression may have because of special properties of the algebra, as we did for an if–then–else expression.

4 A Solution: Classified-model signature generalizations

The technique chosen to solve the problems of MSA is to generalize the notion of a signature and of the sort of an operation. Recall that the sorts of an MSA are not ordered, and that each operation symbol has a unique sort prescribed by the syntactic signature. Polymorphism can be achieved in the CM proposal by allowing an operation symbol to have more than one sort. Subsorts can be achieved by allowing a partial order on the sort symbols to be induced by sort declarations of terms in the assertion language. This is the Classified-Model approach of this paper.

The classified-model (CM) approach to the specification and verification of abstract data types and modules (abstract data types with hidden “state” sorts) allows a semantic rather than syntactic definition of subsorts. A classified model is a single-sorted model which has a classification of its universe into a family of not necessarily disjoint subsets.

The signature section of a conventional MSA or of an order-sorted specification embodies a fundamental assumption of these methods: they are based on a notion of sort which is primarily syntactic. That is, a type system based upon a signature is a classification of syntactic objects, i.e., expressions. The sort of an expression is dependent upon the sorts of its subexpressions.

In the classified model (CM) approach the declarations of a MSA signature are promoted to the status of full-fledged assertions. The information that is coded in a conventional signature, and more, can be expressed as declaration assertions. With this view, a type system is primarily a classification of semantic objects, i.e., of data objects. The sort of an expression can be deduced
as a theorem and is not just a syntactic consequence of the sorts of its subexpressions.

Although the classification permitted by the order-sorted methods is more sophisticated than those allowed by the MSA technique, the implications of the signature remain the same. The CM approach can be viewed as carrying an enrichment from MSA to order-sorted methods to its logical conclusion.

5 Type properties

Declaration assertions allow explicit formulation of the type properties of expressions, which in other systems are formulated implicitly by the signature. The availability of declarations relieves us from the necessity of encoding sort information in a syntactic form (as in an MSA or order-sorted signature) and at the same time is much more general as well. The simplest classified approach would include in the signature just equality and sort predicate symbols, but the generalization to arbitrary predicate symbols does not change the type structure and also has some advantages in the expressibility of specifications. It also makes sense to include this more general signature because all of the same mathematical concepts that are needed to formalize the simplest approach are needed as well for the general approach. The simplest approach was first described by W. W. Wadge [9].

For example, the CM declaration assertion

\[( T_{stk}, I_{int} ) \rightarrow stk( push(T_{stk}, I_{int}) ) \] (6)

replaces the conventional MSA syntactic typing of push as

\[ push : stk, int \rightarrow stk. \] (7)

Polymorphism can be accommodated by using more than one declaration; for example,

\[
\begin{align*}
(X_{nat}, Y_{nat}) & \rightarrow nat(X_{nat} + Y_{nat}) \\
(X_{int}, Y_{int}) & \rightarrow int(X_{int} + Y_{int}) \\
(X_{real}, Y_{real}) & \rightarrow real(X_{real} + Y_{real})
\end{align*}
\]

Declarations can also be used to specify an ordering between sorts; the declaration

\[( X_{int} ) \rightarrow real(X_{int}) \]

asserts that sort int is a subsort of sort real. This declaration is true in a model M with interpretation function \( \alpha \) iff \( \alpha(int) \) is a subset of \( \alpha(real) \).

Declaration assertions can be used to give "special-case" sort information which cannot be deduced from sort information of subterms, for example

\[
\begin{align*}
(X_{real}) & \rightarrow int(X_{real} - X_{real}) \\
(X_{int}, Y_{real}) & \rightarrow int(\text{if true then } X_{int} \text{ else } Y_{real}) \\
(X_{real}) & \rightarrow real(\text{if } X_{real} > 0 \text{ then } \sqrt{X_{real}} \text{ else } \sqrt{-X_{real}})
\end{align*}
\]

6 Syntax

The definition of the language of classified-model specifications (the CM language) is divided into three parts. The vocabulary introduces disjoint sets of symbols which are used to construct the language. The terms and formulas give formulation rules for the construction of legal objects of the language.

The vocabulary consists of symbols that have a fixed meaning and symbols whose meaning is to be defined, also known as a signature. In the applied logic, or object language, used in
specification examples, the symbols of the signature are in the math italics font and the symbols are chosen to convey to the reader the defined meaning. In metatheoretic proofs, such as soundness and completeness, the symbols of the signature are metavariables that range over the symbols of the object language. These metavariables are depicted in sans serif type font.

Definition 1 (Vocabulary). The vocabulary consists of all the defined symbols of the language.

1. The implication connective is $\rightarrow$.
2. A set of signatures is defined, where a specific signature $\Sigma$ contains function and predicate symbols of defined arities, or number of argument places.
   (a) The function symbols include the constant (0-ary function) symbols.
   (b) The predicate symbols include:
      i. the propositional constant (0-ary predicate) symbols,
      ii. the sort (unary predicate) symbols, and
      iii. the infix binary identity predicate “$= = \bot$”, which is different from the metalanguage equality symbol “$=$”.
3. A set of unsorted variable symbols is defined.
4. A set of sorted variable symbols is defined, where each sorted variable symbol is subscripted by some sort predicate symbol.
5. The universal quantifier ($\forall \cdot \, X \cdot$) is defined, where $X$ is any set of variables.

Notation:
1. Script symbols are used to represent:
   (a) an arbitrary formula, e.g., $F$ represents either an atomic formula or a Horn formula (defined below);
   (b) a collection of variables, e.g., $X$.
2. Brackets “[ ]” and “[ ]” are used to enclose CM syntax objects in mathematical expressions.

The set of $\Sigma(\mathcal{X})$-terms is constructed from a signature $\Sigma$ and a set of variables $\mathcal{X}$ by combining function symbols with operand expressions.

Definition 2 ($\Sigma(\mathcal{X})$-Terms). The set of $\Sigma(\mathcal{X})$-terms, denoted $\Sigma(\mathcal{X})$, is defined inductively from a signature $\Sigma$ and a set of variables $\mathcal{X}$ by:

1. every variable symbol in $\mathcal{X}$ is a term;
2. if $t_0, \ldots, t_{n-1}$ are terms and $f$ is an $n$-ary function symbol, then $f(t_0, \ldots, t_{n-1})$ is a term.

The set of ground $\Sigma$-terms, $\Sigma(\emptyset)$, has no variable symbols.

The set of formulas is constructed from the set of terms by using the implication logical connective and universal quantification.

Definition 3 ($\Sigma$-Formulas). The set of $\Sigma$-formulas is defined inductively from a signature $\Sigma$ and a set of variables $\mathcal{X}$ by:

1. if $t_0, \ldots, t_{n-1}$ are $\Sigma(\mathcal{X})$-terms and $P$ is an $n$-ary predicate symbol, then $P(t_0, \ldots, t_{n-1})$ is an atomic formula;
2. if $A$ is an atomic formula (the head) and the set $\{B_0, \ldots, B_{n-1}\}$ ($n \geq 0$) are atomic formulas (the body) then $B_0, \ldots, B_{n-1} \rightarrow A$ is a Horn formula;

Notation:
1. Script symbols are used to represent:
   (a) an arbitrary formula, e.g., $F$ represents either an atomic formula or a Horn formula (defined below);
   (b) a collection of variables, e.g., $X$.
3. if \( \mathcal{X} \) is a list of variable symbols and \( \mathcal{F} \) is an atomic or Horn formula, then \((\mathcal{X})\mathcal{F}\) is a closed formula, where \( \mathcal{X} \) includes (at least) all the variables occurring in all the terms in \( \mathcal{F} \).

A Horn formula with no body is identified with an atomic formula having just the head, but not the implication connective.

Whenever “formula” is mentioned below, it is assumed to be closed unless otherwise specified.

**Definition 4** (Substitution). Let \( \Sigma \) be a signature, let \( \mathcal{X} \) and \( \mathcal{Y} \) be sets of variables and let \( \theta \) be a function \( \theta : \mathcal{X} \rightarrow \Sigma(\mathcal{Y}) \). \( \theta \) is called a substitution function and can be extended to a function \( \theta^* : \Sigma(\mathcal{X}) \rightarrow \Sigma(\mathcal{Y}) \):

1. for each variable symbol \( X \) in \( \mathcal{X} \), \( \theta^* X = \theta X \);
2. for each \( n \)-ary function symbol \( f \), \( \theta^* f(t_0, \ldots, t_{n-1}) = f(\theta^* t_0, \ldots, \theta^* t_{n-1}) \);
3. for each \( n \)-ary predicate symbol \( P \), \( \theta^* P(t_0, \ldots, t_{n-1}) = P(\theta^* t_0, \ldots, \theta^* t_{n-1}) \).

\( \theta^* \), commonly abbreviated as \( \theta \), can also be applied to a Horn formula by applying it to each individual atomic formula.

**Definition 5** (Specification). A specification \( S = (\Sigma, \Gamma) \) consists of a signature \( \Sigma \) and a set of closed \( \Sigma \)-formulas \( \Gamma \).

The following terminology is used:

1. An **equation** is an atomic formula constructed from just the equality predicate.
2. A **conditional equation** is a Horn formula constructed from just the equality predicate.
3. A **base assertion** is an atomic formula constructed from a unary predicate symbol.
4. A **generation assertion** is a Horn formula having a unary predicate symbol in the head.
5. A **declaration assertion** is a base or generation assertion.
6. A **declaration** is an instance of a unary sort predicate.

### 7 Semantics

The semantics of the CM language provides a meaning for the syntax objects described above by defining:

1. \( \Sigma \)-model for the elements of the signature;
2. assignment function for variables;
3. extended assignment function for terms;
4. truth in a model for closed formulas.

**Definition 6** (\( \Sigma \)-Model). For any given signature \( \Sigma \), a \( \Sigma \)-model \( \mathcal{M} \) is a pair \((D_\mathcal{M}, \alpha_\mathcal{M})\) where

1. \( D_\mathcal{M} \) is a non-empty set called the universe, or domain, of \( \mathcal{M} \);
2. \( \alpha_\mathcal{M} \) is a \( \Sigma \)-interpretation function which assigns:
   (a) for each 0-ary function (i.e., constant) symbol \( c \) in \( \Sigma \): \( \alpha_\mathcal{M}[c] \in D_\mathcal{M} \);
   (b) for each other \( n \)-ary function symbol \( f \) in \( \Sigma \) a function \( \alpha_\mathcal{M}[f] : (D_\mathcal{M})^n \rightarrow D_\mathcal{M} \);
   (c) for each 0-ary function (i.e., propositional) symbol \( P \) in \( \Sigma \) an element \( \alpha_\mathcal{M}[P] \in \{T,F\} \);
(d) for each other n-ary predicate symbol $P$ in $\Sigma$ a subset $\alpha_M[P]$ of the Cartesian power $(D_M)^n$. The truth set of the equality predicate must be $\{(d, d) : d \in D_M\}$.

Like a conventional single-sorted model, there is no a priori notion of sort for operation symbols. The sets $\{\alpha(s) : s$ is a sort symbol$\}$ correspond to the carriers in a conventional MSA. In a classified model $M$ there might be elements of the universe which are not in any of the “carriers”. These are error objects which are the result of operations such as pop(stknil) that have no intended purpose. It is shown below that for “normal” specifications in which every term in the specification is of the form $\theta$, there might be elements of the universe which are not in any of the “carriers”. These are extraneous objects caused by no difficulties in practice.

A particular $\Sigma(X)$-term model is defined by generation over a set of variable symbols.

**Definition 7 ($\Sigma(X)$-term model $T_\Sigma(X)$).** Let $\Sigma$ be a signature and $X$ a set of variables, then $T_\Sigma(X)$ is the $\Sigma$-model with universe the set of $\Sigma(X)$-terms and an interpretation mapping $\alpha$ for the symbols in $\Sigma$ such that:

1. each $\Sigma(X)$-term is interpreted as itself, that is:
   
   (a) for each variable $X \in X : \alpha[X] = X$
   
   (b) If $t_0, \ldots, t_{n-1}$ are $\Sigma(X)$-terms and $f$ is an n-ary function symbol, then
   
   $$\alpha[f](\alpha[t_0], \ldots, \alpha[t_{n-1}]) = f(t_0, \ldots, t_{n-1})$$

2. for the identity predicate symbol: $\alpha[\equiv] = \{(t, t) : t$ is a $\Sigma(X)$-term$\}$;

3. any sort predicate symbol $s$ is interpreted as just the set of sorted variables that have that sort subscript, i.e., $\alpha[s] = \{X_s : X_s$ is a variable with sort subscript $s\}$;

4. any other predicate symbol $P$ is interpreted as the empty set, i.e., $\alpha[P] = \{}$;

$T_\Sigma(\emptyset)$ is abbreviated $T_\Sigma$.

Any $\Sigma$-model is given a $\Sigma(X)$-model structure by extending a $\Sigma$-interpretation by an assignment function which interprets the variable symbols.

**Definition 8 (Assignment function).** Let $\Sigma$ be a signature, let $X$ be a set of variables, let $M = (D_M, \alpha)$ be a $\Sigma$-model with interpretation function $\alpha$ and let $\theta$ be a function $\theta : X \to D_M$ ($\theta$ is called an assignment function). The extended assignment function $\theta^*$ interprets $\Sigma(X)$-terms and formulas.

1. for each variable symbol $X \in X$, $\theta^*[X] = \theta[X]$.
2. for each constant symbol $c$, $\theta^*[c] = \alpha[c]$.
3. for each function symbol $f$, $\theta^*[f(t_0, \ldots, t_{n-1})] = \alpha[f](\theta^*[t_0], \ldots, \theta^*[t_{n-1}])$.
4. for each propositional symbol $P$, $\theta^*[P] = \alpha[P]$.
5. for terms $t_1$ and $t_2$,

$$\theta^*[t_1 = t_2] = \begin{cases} T, & \theta^*[t_1] = \theta^*[t_2] \\ F, & \text{otherwise.} \end{cases}$$

6. for each predicate symbol $P$,

$$\theta^*[P(t_0, \ldots, t_{n-1})] = \begin{cases} T, & (\theta^*[t_0], \ldots, \theta^*[t_{n-1}]) \in \alpha[P] \\ F, & \text{otherwise.} \end{cases}$$
7. for each Horn formula $B_0, \ldots, B_{m-1} \rightarrow A,$

$$\theta^*[B_0, \ldots, B_{m-1} \rightarrow A] = \begin{cases} T, & \text{if } \theta^*[B_i] = T, i = 0..m-1, \theta^*[A] = T \\ F, & \text{otherwise.} \end{cases}$$

**Definition 9** (Sorted Assignment). An assignment function $\theta: X \rightarrow D_M$ is a sorted assignment function if whenever $X_i \in X$ then $\theta[X_i] \in \alpha[s]$, and similarly for an extended sorted assignment.

**Definition 10** (Truth in a Model). The truth value of a closed formula $(X) F$ in a model $M$ with universe $D_M$ and interpretation function $\alpha$ is defined by extending $\alpha$ to a function $\alpha^*$:

$$\alpha^*[(X) F] = \begin{cases} T, & \text{if } \theta^*[F] = T \text{ for any sorted assignment } \theta: X \rightarrow D_M \\ F, & \text{otherwise} \end{cases}$$

**Definition 11** ($S$-model, $\models$). Let $\Sigma$ be a signature and $\Gamma$ a set of $\Sigma$-formulas. For a specification $S = (\Sigma, \Gamma)$, a formula $(X) F$ and any $\Sigma$-model $M$ with interpretation mapping $\alpha$:

1. $M \models (X) F$ (M satisfies $(X) F$, or $(X) F$ is valid in $M$) iff $\alpha^*[(X) F] = T$;
2. $S$-model: a $\Sigma$-model that satisfies all formulas of $\Gamma$;
3. $\text{Mod}(S)$: the class of all $S$-models;
4. $\text{Th}(M)$ (theory of $M$): the set of all $\Sigma$-formulas that are valid in $M$;
5. $S \models (X) F$ ($(X) F$ is a logical consequence of $S$) iff $(X) F$ is satisfied in any $S$-model.
6. $\Gamma \models F$ is an abbreviation for $S \models F$ when $\Sigma$ is obvious from the context of $S = (\Sigma, \Gamma)$.

Along with the definition of model, it is useful to define a homomorphism between models as a structure-preserving mapping.

**Definition 12** ($\Sigma$-Homomorphism). Let $M$ and $N$ be $\Sigma$-models with interpretation mappings $\alpha$ and $\beta$, then $h: D_M \rightarrow D_N$ is a $\Sigma$-homomorphism if:

1. $h(\alpha[f](d_0, \ldots, d_{n-1})) = \beta[f](h(d_0), \ldots, h(d_{n-1}))$ for every $n$-ary function symbol $f$ and all $d_0, \ldots, d_{n-1} \in D_M$.
2. if $\langle d_0, \ldots, d_{n-1} \rangle \in \alpha[P]$ then $\langle h(d_0), \ldots, h(d_{n-1}) \rangle \in \beta[P]$ for every $n$-ary predicate symbol $P$ and all $d_0, \ldots, d_{n-1} \in D_M$.

### 8 Initial models

The specification of abstract data types and modules, a generalization of abstract data types in which some sorts are designated as “hidden state” sorts, was the primary motivation for the development of the classified approach. A specification is a collection of assertions which is satisfied by the models of the intended data types and operations. In general, a specification has many different models. That is, there are many different models in which all the assertions in the specification are true. The set of initial term models of the specification is one particular collection of models associated with the specification which have a claim to be the intended standard models.

An initial term model, defined below, can also be characterized as a model which is:

**Generated.** The objects we specify must have names. Each object in any generated model is represented by some term of the signature.

**Generic.** The model is the most general possible in the sense that any other model is a special case of a generic model, e.g., only those objects are identified which the specification requires to be identified and only those objects are classified as sort $s$ which the specification requires to be so classified.
The initial model is therefore the minimal model to the specification in the sense that the truth set of each predicate symbol (including \( = = \)) is minimal.

**Definition 13 (Initial Model).** Let \( \mathcal{C} \) be a class of \( \Sigma \)-models. A \( \Sigma \)-model \( \mathcal{I} \in \mathcal{C} \) is called initial in \( \mathcal{C} \) iff for each \( \Sigma \)-model \( \mathcal{M} \in \mathcal{C} \) there is a unique \( \Sigma \)-homomorphism \( h : \mathcal{I} \to \mathcal{M} \).

If \( \mathcal{C} \) is of the form \( \text{Mod}(\Gamma) \), where \( \Gamma \) is some set of \( \Sigma \)-formulas, we also say that \( \mathcal{I} \) is initial for \( \Gamma \).

A generalization of initiality is given by freeness:

**Definition 14 (Free Model).** Let \( \mathcal{C} \) be a class of \( \Sigma \)-models, \( X \) a set of variables and \( \mathcal{M} \in \mathcal{C} \). A \( \Sigma \)-model \( \mathcal{F} \in \mathcal{C} \) is called free over \( X \) in \( \mathcal{C} \) iff there is a sorted assignment \( u : X \to D_F \), called a universal mapping, such that for every sorted assignment \( \theta : X \to D_M \) there is a unique homomorphism \( \theta' : D_F \to D_M \) such that \( \theta = \theta' \circ u \).

Like the MSA and order-sorted systems, there is always an initial model in the class of models satisfying a set of Horn formulas [7]. Furthermore, it is also shown in [7] that any (finite) first-order specification that admits an initial model is equivalent to a (finite) Horn specification. For a specification \( S \) in a language \( L \) to admit an initial model means that \( S \) has an initial model and for any set of assertions \( C \) of \( L \), the specification \( S \cup C \) also has an initial model, but not necessarily the same initial model. This is an important characteristic for program specifications which evolve through enrichment by additional assertions or are parameterized by other specifications to be included later. Therefore, if it is important to ensure that a specification has an initial model then it is fruitless to search for first-order specification languages more powerful than Horn specifications.

### 9 Classified model deduction

In the CM method sorts are declared by assertions, whereas in the order-sorted method sorts are declared in a syntactic sort declaration section that is separate from the assertions. The semantic consequence of this is that classified models may have “error” objects that are not in the truth set of any sort predicate, e.g., \( \text{pop}(\text{stknil}) \) in the stack example. These objects do not even exist in order-sorted models but they are rendered harmless in the classified-model system since the rules of inference described below introduce only classified terms into a proof derived from a specification having only classified terms, i.e., a “normal specification”.

**Definition 15 (Classified \( \Sigma \)-Term).** Let \( S = (\Sigma, \Gamma) \) be a specification and let \( t \) be a \( \Sigma(X) \)-term, then \( t \) is a classified \( \Sigma \)-term iff in the initial \( \Sigma \)-model with interpretation \( \alpha \) and for all sorted assignment functions \( \theta \) that extend \( \alpha \) there is a classification \( s \in \Sigma \) (perhaps descending upon \( \theta \)) such that:

\[
\theta^*(t) \in \alpha(s). \tag{8}
\]

**Definition 16 (Normal specification).** \( S = (\Sigma, \Gamma) \) is a normal specification iff each term \( t \) appearing in \( \Gamma \) is a classified \( \Sigma \)-term.

A variant of the order-sorted model rules of [3] is sound and complete for atomic formulas of the CM system. Even incompletely specified operations are allowed because from a normal specification the rules allow the introduction of only classified terms. Incompletely specified operations are not the same as partial functions because in the initial model, and also in the free model, every term of the language has a meaning. The classification, or lack of classification, does not affect the existence of a defined meaning for each term of the language.
The set of derivable formulas is defined by the following set of rules of inference. Each formula below is universally quantified, where all the variables appearing in such a formula are represented in the list of variables.

1. **Reflexivity.** $(X) t = t$ is derivable.

2. **Symmetry.** If $(X) t = t'$ is derivable then $(X) t' = t$ is derivable.

3. **Transitivity.** If $(X') t = t'$ and $(X'') t' = t''$ are derivable and if $X = X' \cup X''$ then $(X) t = t''$ is derivable.

4. **Substitutivity.** If $(X_i) t_i = t_i'$ is derivable and if $X = \cup X_i, i = 0..n - 1$, then:
   
   (a) for any $n$-ary function symbol $f$ in $\Sigma$ the equation
      
      $$(X) f(t_0, \ldots, t_{n-1}) = f(t_0', \ldots, t'_{n-1})$$
      
      is derivable;
   
   (b) for any $n$-ary predicate symbol $P$ in $\Sigma$, if $(X) P(t_0, \ldots, t_{n-1})$ is derivable then
      
      $$(X) P(t_0', \ldots, t'_{n-1})$$
      
      is derivable.

5. **Modus ponens.** If $(X) B_0, \ldots, B_{m-1} \rightarrow A$ is derivable and if $\theta : X \rightarrow \Sigma(Y)$ is any substitution such that:
   
   (a) each $(Y) \theta B_i$ is derivable for $i = 0..m - 1$, and
   
   (b) $(Y)s(\theta X_s)$ is derivable for each $X_s \in X$,

   then $(Y) \theta A$ is derivable.

The substitution in Rule 5 of a set of terms $T_{\Sigma}(Y)$ for a set of variables $X$ includes the notion of change of variables, as well as the elimination and introduction of variables. In such a substitution a term of $T_{\Sigma}(Y)$ must be provably of sort $s$ if it is substituted for a variable $X_s$ of sort $s$. This substitution reflects the fact that it is important to include explicit quantifiers in any specification since any variable can only be eliminated by replacing it with some term which is provably of the same sort. It is this rule that in a derivation from a normal specification restricts the introduction of terms to those that are provably of the same classification.

For example, Rule 4b is illustrated with the equation

$$(X'_{int}) \times X'_{int} = X'_{int} \times 2 \quad (9)$$

and the assertion

$$(X_{even}, X'_{int}) \ even(X_{even} + (2 \times X'_{int})) \quad (10)$$

to infer

$$(X_{even}, X'_{int}) \ even(X_{even} + (X'_{int} \times 2)) \quad (11)$$

Also, Rule 5 is illustrated by using the assertion

$$(X_{pos}, X'_{pos}) \ pos(X_{pos} + X'_{pos}) \quad (12)$$

and the substitution $\theta : \{(X_{pos}, Y_{pos}), (X'_{pos}, Y'_{int} \times X'_{int})\}$, where the sort of $\theta X'_{pos}$ is assured by the assertion

$$(Y'_{int}) \ pos(Y'_{int} \times Y'_{int}) \quad (13)$$

11
to derive

\[(Y_{\text{pos}}, Y'_{\text{int}}) \text{ pos}(Y_{\text{pos}} + (Y'_{\text{int}} \times Y'_{\text{int}}))\]  \hspace{1cm} (14)

These rules can also be applied to more general formulas.

Let \( S = (\Sigma, \Gamma) \) be a specification with assertions \( \Gamma \) and suppose that we are interested in proving certain properties of the data types specified. Any assertions which can be derived using the rules of inference just given are true in all models. These are non-ground assertions which are true in the initial model but which are not true in other models of the specification and so cannot be derived using the ordinary rules of inference. For example, the commutative law of addition is not a consequence (via the five rules) of the specification of the natural numbers given earlier.

An induction rule is a stronger rule of inference which takes into account the special features of the initial model. The universe of the initial model contains only those data objects required to exist by the specification. In the CM system we can state that the classified elements of the initial model are exactly those which are generated by the declaration assertions in the specification. Therefore we can state a rule allowing assertions to be proved by induction on the complexity of the structure of the data objects of a given type.

Suppose that \( \Sigma \) is a signature, that \( S = (\Sigma, \Gamma) \) is a specification, that \( s \in \Sigma \) is a sort symbol that appears in the assertions \( S \) and that \( P(X_s) \) is an atomic assertion involving the variable \( X_s \) of sort \( s \) (and possibly others, possibly of other sorts, as well). Let \( A_i \rightarrow s(t_i), i = 0..n - 1 \), be all the declaration assertions for the sort \( s \) in \( S \), where the body \( A_i \) of each declaration assertion is either empty or a set of atomic assertions.

1. Form a new sequence \( t'_0, \ldots, t'_{n-1} \) of terms in which each \( t'_i \) is the result of replacing all variables of sort \( s \) in \( t_i \) by new constant symbols \( c_0, \ldots, c_{m-1} \) not already in the signature \( \Sigma \).
2. Prove, using the ordinary rules of inference, the assertions \( P(t'_i) \) for \( i < n \). In doing so we use \( S \), the declarations \( s(c_j), j < m \), and the induction hypotheses \( P(c_j), j < m \). Having done so, we can conclude that \( P(X_s) \) is true in the initial model of \( S \).

One of the advantages of this induction rule is that it allows proofs to be formulated entirely within the object language, i.e., essentially that used by programmers. It does not require knowledge of, or reference to, metamathematical notions such as that of a homomorphism.

10 Free \( \Sigma \)-term model

**Theorem 1** (Free \( \Sigma \)-term model). The \( \Sigma \)-term model \( T_\Sigma(\mathcal{X}) \) is a free \( \Sigma \)-model over \( \mathcal{X} \) in the class of \( \Sigma \)-models.

**Proof.** Let the inclusion \( u : \mathcal{X} \rightarrow T_\Sigma(\mathcal{X}) \) (a sorted assignment) be the universal mapping of Definition 14. Let \( M \) be a \( \Sigma \)-model with interpretation \( \alpha \). It must be shown that for any sorted assignment \( \theta : \mathcal{X} \rightarrow D_M \) that the extended sorted assignment \( \theta^* : D_{T_\Sigma(\mathcal{X})} \rightarrow D_M \) is the required unique \( \Sigma \)-homomorphism such that \( \theta = \theta^* \circ u \).

\[ T_\Sigma(\mathcal{X}) \xrightarrow{u} \mathcal{X} \]
\[ \theta^* \downarrow \theta \]
\[ M \]

\( \theta^* \) satisfies the requirements of a homomorphism (Definition 12):

1. For function symbols, Definition 8 for \( \theta^* \) is applied directly.
2. For predicate symbols, there are three cases:
   (a) The identity predicate case can be shown easily by structural induction over terms.
(b) Any sort symbol $s$ has an interpretation in the term model $T_{\Sigma}(\mathcal{A})$ as the set of variables with that subscript. Since $\theta^*[X_s] = \theta[X_s]$ and $\theta$ is a sorted assignment, the condition for a homomorphism is satisfied.

(c) Any other predicate symbol has an empty truth set in the term model, so the condition for a homomorphism is satisfied vacuously.

To show that $\theta^*$ is the unique $\Sigma$-homomorphism which extends $\theta$, assume there is another $\Sigma$-homomorphism

$$\gamma : D_{T_{\Sigma}(\mathcal{A})} \to D_M$$

such that $\theta = \gamma \circ \alpha$. It is shown below by structural induction that $\gamma = \theta^*$.

1. Base case for variables: $\theta^*[X_s] = \theta[X_s]$ by assumption for all variable symbols $X_s$, since $\gamma$ extends $\theta$.

2. Induction case: Assume $f(t_0, \ldots, t_{n-1}) \in D_{T_{\Sigma}(\mathcal{A})}$ and $\theta^*[t_i] = \gamma[t_i]$, $i = 0..n - 1$.

(a) $\theta^*[f(t_0, \ldots, t_{n-1})] = \alpha[f](\theta^*[t_0], \ldots, \theta^*[t_{n-1}])$ 
(b) $= \alpha[f](\gamma[t_0], \ldots, \gamma[t_{n-1}])$ 
(c) $= \gamma[f(t_0, \ldots, t_{n-1})]$ 

Defn. 12 (homo.)

Assumption

□

11 Quotient models

Quotient models are important because they are used in the construction of a free model of a specification. The following definitions, which support the free model construction of the next section, are either standard or are minor variations of standard definitions.

Definition 17 (Equivalence Relation). A relation $R$ on a set $D$ is called an equivalence relation on $D$ if $R$ is reflexive, symmetric and transitive.

Definition 18 (Congruence Relation). Let $\Sigma$ be a signature and $M$ a $\Sigma$-model with universe $D_M$ and interpretation function $\alpha$, then an equivalence relation $R$ on $D_M$ is called a congruence relation if for each $d_i, d'_i \in D_M$ such that $(d_i, d'_i) \in R$ for $0 \leq i < n$, the following conditions hold:

1. for each $n$-ary function symbol $f$, 

$$(\alpha[f](d_0, \ldots, d_{n-1}), \alpha[f](d'_0, \ldots, d'_{n-1})) \in R.$$ 

2. for each $n$-ary predicate symbol $P$:

$$\langle d_0, \ldots, d_{n-1} \rangle \in \alpha[P] \iff \langle d'_0, \ldots, d'_{n-1} \rangle \in \alpha[P].$$

Definition 19 (Congruence Class). Let $\Sigma$ be a signature, $M$ be a $\Sigma$-model and $R$ be a congruence relation on $D_M$. A congruence class $[a]$ of $R$ is a subset of $D_M \times D_M : [a] = \{ b : (b, a) \in R \}$.

Definition 20 (Kernel). Let $M$ and $N$ be $\Sigma$-models and let $f : M \to N$ be a $\Sigma$-homomorphism. The kernel of $f$, $\ker(f)$, is the set

$$\{(a, b) : f(a) = f(b)\}. \quad (16)$$

Note that $\ker(f)$ is not necessarily a congruence relation on $M$, a principal difference between this work and the equational case.
**Definition 21** (Quotient Model $M_R$). Let $M$ be a $\Sigma$-model with universe $D_M$ and interpretation function $\alpha$. Let $R$ be a congruence relation on $D_M$ and let $d_i \in D_M$, $i = 0..n-1$. The quotient model $M_R$ has as universe the set of $R$-congruence classes and interpretation function $\beta$ such that:

1. for each $n$-ary function symbol $f$:
   \[
   \beta[f]([d_0], \ldots, [d_{n-1}]) = [\alpha[f](d_0, \ldots, d_{n-1})]
   \]

2. for each $n$-ary predicate symbol $P$:
   \[
   ([d_0], \ldots, [d_{n-1}]) \in \beta[P] \iff (d_0, \ldots, d_{n-1}) \in \alpha[P].
   \]

The quotient model is well defined since the relation $R$ is a congruence.

**Theorem 2** (Universal property of quotients). Let $\Sigma$ be a signature, $M$ a $\Sigma$-model and $R$ a congruence relation on $M$. Then $q : M \rightarrow M_R$ defined by $q(a) = [a]$, for $a \in M$, is a $\Sigma$-homomorphism (the quotient homomorphism) and satisfies the following universal property: Let $N$ be a $\Sigma$-model and $f : M \rightarrow N$ be any homomorphism such that $R \subseteq \ker(f)$, then there is a unique $\Sigma$-homomorphism $f' : M_R \rightarrow N$ such that $f = f' \circ q$.

\[
\begin{array}{c}
M_R = (D_{M_R}, \beta) \\
\overset{q}{\rightarrow} M = (D_M, \alpha) \\
\overset{f}{\longrightarrow} N = (D_N, \gamma)
\end{array}
\]

**Proof.** The mapping $q$ is a $\Sigma$-homomorphism since Definition 21 satisfies the homomorphism criteria in Definition 12.

To prove the universal property of $q$ define $f' : M_R \rightarrow N$ by $f'([a]) = f(a)$ for all $a \in M$. Uniqueness is easily shown by observing that for any other $\Sigma$-homomorphism $f'' : M_R \rightarrow N$ with $f = f'' \circ q$ it must be the case that $f''([a]) = f(a)$ for all $a \in M$.

First, $f'$ is well defined:

1. $[a] = [b]$ \hspace{1cm} Assumption
2. $(a, b) \in R$ \hspace{1cm} 1, Defn. 19 (congruence class)
3. $(a, b) \in \ker(f)$ \hspace{1cm} 2, $R \subseteq \ker(f)$
4. $f(a) = f(b)$ \hspace{1cm} 3, Defn. 20 (kernel)
5. $f'([a]) = f'([b])$ \hspace{1cm} 4, Defn. of $f'$.

Second, $f'$ is a $\Sigma$-homomorphism. Let $\alpha$ be the interpretation for $M$, $\beta$ for $M_R$ and $\gamma$ for $N$. For $g$ an $n$-ary $\Sigma$-function symbol and $[\alpha[t_i]] \in D_{M_R}$, $i = 0..n-1$,

1. $f'([\beta[g]([\beta[t_0]], \ldots, \beta[t_{n-1}]])) = f'([\alpha[g]([\alpha[t_0]], \ldots, [\alpha[t_{n-1}]])])$ \hspace{1cm} Defn. of $q$ homo.
2. $= f'([\alpha[g]([\alpha[t_0]], \ldots, [\alpha[t_{n-1}]])])$ \hspace{1cm} Defn. of $q$ homo.
3. $= f(\alpha[g]([\alpha[t_0]], \ldots, [\alpha[t_{n-1}]]))$ \hspace{1cm} Defn. of $f'$
4. $= \gamma[g]((f(\alpha[t_0]), \ldots, f(\alpha[t_{n-1}])))$ \hspace{1cm} $f$ homo.
5. $= \gamma[g]((f'([\alpha[t_0]]), \ldots, f'([\alpha[t_{n-1}]])))$ \hspace{1cm} Defn of $f'$
6. $= \gamma[g]((f'([\beta[t_0]]), \ldots, f'([\beta[t_{n-1}]])))$ \hspace{1cm} Defn of $q$. 

For $P$ an $n$-ary $\Sigma$-predicate symbol and $[\alpha[t_i]] \in D_{\Sigma}, i = 0..n - 1$:

1. $\langle \beta[t_0], \ldots, \beta[t_{n-1}] \rangle \in \beta[P]$  
   Assumption

2. $\langle [\alpha[t_0]], \ldots, [\alpha[t_{n-1}] \rangle \in \beta[P]$  
   Defn. of $q$

3. $\langle \alpha[t_0], \ldots, \alpha[t_{n-1}] \rangle \in \alpha[P]$  
   Defn. of quotient

4. $\langle f(\alpha[t_0]), \ldots, f(\alpha[t_{n-1}] \rangle \in \gamma(P)$  
   $f$ homo.

5. $\langle f'(\alpha[t_0]), \ldots, f'(\alpha[t_{n-1}] \rangle \in \gamma(P)$  
   Defn. of $f'$

6. $\langle f'(\beta[t_0]), \ldots, f'(\beta[t_{n-1}] \rangle \in \gamma(P)$  
   Defn. of $q$.  

$\square$

12 Proof theory

The development of this proof theory parallels the Order-Sorted-Model proof theory of [3].

12.1 Existence of the $S$-model $\Sigma_S(X)$

The free model for a specification $S = (\Sigma, \Gamma)$ is based upon a congruence $\equiv_S$ on $\Sigma(X)$ terms. This congruence is generated by the assertions $\Gamma$ and, because it is based on provability using the rules of inference, results in a simple proof of completeness for these rules. This is the same congruence and free model construction used in [3].

**Theorem 3** (Congruence $\equiv_S$). Let $S = (\Sigma, \Gamma)$ be a specification and let $t$ and $t'$ be $\Sigma(X)$-terms, then the property

$$(X) \ t \equiv_S t' \text{ iff } t \equiv_S t' \text{ is derivable from } \Gamma \text{ using Rules of inference 1–5}$$

defines a congruence relation $\equiv_S$ on $\Sigma(X)$-terms.

**Proof.** The first three rules of inference define an equivalence relation and the fourth ensures that the property defines a congruence relation.  

$\square$

**Definition 22** (Quotient Term Model $\mathcal{T}_S(X)/\equiv_S$). Let $S = (\Sigma, \Gamma)$ be a specification and let $X$ be the set of variables in $S$, then the quotient term model is $\mathcal{T}_S(X)/\equiv_S$.

By Definition 7 for $\mathcal{T}_S(X)$ and Definition 21 for a quotient model, $\mathcal{T}_S(X)/\equiv_S$ has as universe the set of congruence classes and an interpretation function $\alpha$ such that

1. for the identity predicate symbol:

   $$\alpha[\cdot] = \{ \{d, d\} : d \text{ is a congruence class of } \equiv_S \};$$

2. any sort predicate symbol $s$ is interpreted as just the congruence classes of the sorted variables that have that sort subscript, i.e.,

   $$\alpha[s] = \{ [X_s] : X_s \text{ is a variable with sort subscript } s \};$$

3. any other predicate symbol $P$ is interpreted as the empty set, i.e., $\alpha[P] = \{ \}$.

**Definition 23** (Specification Term Model $\mathcal{T}_S(X)$). Let $S = (\Sigma, \Gamma)$ be a specification and let $X$ be the set of variables in $S$, then the specification term model $\mathcal{T}_S(X)$ is just the quotient term model, except that the interpretation function $\alpha$ gives an $S$-model structure for each $n$-ary predicate symbol $P$ in $\Sigma$:

$$\alpha(P) = \{ \langle t_0, \ldots, t_{n-1} \rangle : \langle X \rangle P(t_0, \ldots, t_{n-1}) \text{ is derivable from } \Gamma \text{ using Rules of inference 1–5. } \}$$
The specification term model $\mathcal{T}_S(\mathcal{X})$ is well defined because the relation generated by provability using these rules of inference in which (by definition using Rule 4) any choice of representatives for the terms of the quotient model will suffice in the interpretation. That is, by Rule 4 the definition is independent of the representatives $t_i$.

The following theorem shows that the quotient term model defined above also satisfies the specification $S = (\Sigma, \Gamma)$, i.e., it is a $S$-model.

**Theorem 4 (S-model $\mathcal{T}_S(\mathcal{X})$).** $\mathcal{T}_S(\mathcal{X})$ is a $S$-model.

**Proof.** Let $S = (\Sigma, \Gamma)$ be a specification and let $(\mathcal{Y}) \mathcal{B}_0, \ldots, \mathcal{B}_{m-1} \rightarrow A$ be an assertion in $\Gamma$, where

1. $A$ is of the form $Q(t_0, \ldots, t_{n-1})$,
2. $\mathcal{B}_i$ is of the form $P_i(t'_0, \ldots, t'_{n-1})$, for all $0 \leq i < m$.

Assume a specification term model $\mathcal{T}_S(\mathcal{X})$ as defined above with interpretation function $\alpha$.

It is required to show $\alpha^*[(\mathcal{Y}) \mathcal{B}_0, \ldots, \mathcal{B}_{m-1} \rightarrow A] = T$. This is done, according to Definition 10 (Truth in a model), by selecting an arbitrary sorted assignment function $\theta_0$ and showing, using Definition 8 (Assignment function), that any ground instance of the assertion is true in the model $\mathcal{T}_S(\mathcal{X})$.

Let $\theta_0 : \mathcal{Y} \rightarrow D_{\mathcal{T}_S(\mathcal{X})}$ be an arbitrary sorted assignment function sending $Y_s$ in $\mathcal{Y}$ to $[t] \in D_{\mathcal{T}_S(\mathcal{X})}$. By Definition 22 we can choose a representative $t \in D_{\mathcal{T}_S(\mathcal{X})}$ for each $[t] = \theta_0(Y_s)$ such that this function can be factored as:

$$\theta_0 = q \circ \theta$$

where

1. $\theta : \mathcal{Y} \rightarrow D_{\mathcal{T}_S(\mathcal{X})}$ is a substitution;
2. $q : D_{\mathcal{T}_S(\mathcal{X})} \rightarrow D_{\mathcal{T}_S(\mathcal{Y})}$ is the quotient homomorphism that sends $t$ to $[t]$ and is extended in Definition 23;
3. By Definition 23, $(\mathcal{X}) s(\theta Y_s)$ is derivable from $\Gamma$ for each $Y_s \in S$ since $\theta_0$ is a sorted assignment. Similarly for the extension $\theta^*$ of $\theta$; $(\mathcal{X}) s(\theta^* Y_s)$ is derivable.

Since, by Theorem 1, $\mathcal{T}_S(\mathcal{Y})$ is the free $\Sigma$-model over $\mathcal{Y}$ in the class of all $\Sigma$-models there is only one homomorphism from $\mathcal{T}_S(\mathcal{Y})$ to any $\Sigma$-model, i.e., $\theta_0^* = q \circ \theta^*$.

Since the conditions of Definition 10 for truth in a model are satisfied:

$$\alpha^*[(\mathcal{Y}) \mathcal{B}_0, \ldots, \mathcal{B}_{m-1} \rightarrow A] = T.$$  

(18)
12.2 Soundness and completeness of the rules of inference

**Theorem 5 (Soundness).** Any formula \((\mathcal{X})\) \(P(t_0, \ldots, t_{n-1})\) that is derivable from a specification \(S\) by Rules 1–5 is satisfied in all \(S\)-models.

**Proof.** To prove the soundness of each rule it is sufficient to show that the validity of the formulas in the premise of the rule implies the validity of the formulas in the conclusion of the rule, i.e., an untrue conclusion cannot be derived from a true premise. Let:

- \(\Sigma\) be a signature and let \(M\) be an arbitrary \(\Sigma\)-model with universe \(D_M\) and interpretation function \(\alpha\).
- \(\theta\) be an arbitrary sorted assignment \(\theta : \mathcal{X} \to D_M\).

The proof of soundness of each rule follows the statement of the rule.

**Reflexivity.** \((\mathcal{X}) t \equiv t\) is derivable. By induction on the structure of terms:

1. Base case: \((\mathcal{X}) X_s \equiv X_s\)
   - \(a\) for all \(d \in D_M\), \(\langle d, d \rangle \in \alpha[\equiv]\) \(\Sigma\)-model Defn. 6
   - \(b\) \(\langle \theta [X_s], \theta [X_s] \rangle \in \alpha[\equiv]\) 1a, assignment Defn. 8
   - \(c\) \(\alpha^* [X_s] X_s \equiv X_s = T\) 1b, truth for quantification Defn. 10

2. Induction case: Assume \(\alpha^* [\mathcal{X}] t_i \equiv t_i = T\) for \(i = 1..n - 1\) and prove
   \[\alpha^* [\mathcal{X}] f(t_0, \ldots, t_{n-1}) \equiv f(t_0, \ldots, t_{n-1}) = T,\]

   where \(\mathcal{X} = \cup X_i\) and \(f\) is an \(n\)-ary function symbol.

   - \(a\) \(\theta^*[t_i] = \theta^*[t_i], i = 0..n - 1\) Assumption by Defn. 10, Defn. 8
   - \(b\) \(\alpha[f] (\theta^*[t_0], \ldots, \theta^*[t_{n-1}]) = \alpha[f] (\theta^*[t_0], \ldots, \theta^*[t_{n-1}])\) 2a, \(D_M = \)
   - \(c\) \(\theta^*[f(t_0, \ldots, t_{n-1})] = \theta^*[f(t_0, \ldots, t_{n-1})]\) 2b, assignment Defn. 8
   - \(d\) \(\theta^*[f(t_0, \ldots, t_{n-1})] = f(t_0, \ldots, t_{n-1}) = T\) 2c, assignment Defn. 8
   - \(e\) \(\alpha^* [\mathcal{X}] f(t_0, \ldots, t_{n-1}) \equiv f(t_0, \ldots, t_{n-1}) = T\) 2d, Defn. 10.

**Symmetry.** If \((\mathcal{X}) t \equiv t'\) is derivable then \((\mathcal{X}) t' \equiv t\) is derivable.

1. Assume \(\alpha^* [\mathcal{X}] t \equiv t'\)
2. \(\theta^*[t] = t'\) 1, truth for quantification Defn. 10
3. \(\theta^*[t'] = \theta^*[t']\) 2, assignment Defn. 8
4. \(\theta^*[t'] = \theta^*[t']\) 3, symmetry in \(D_M\)
5. \(\theta^*[t'] = \theta^*[t']\) 4, assignment Defn. 8
6. \(\alpha^* [\mathcal{X}] t' \equiv t\) 5, truth for quantification Defn. 10.

**Transitivity.** If \((\mathcal{X'}) t \equiv t'\) and \((\mathcal{X''}) t' \equiv t''\) are derivable and if \(\mathcal{X} = \mathcal{X'} \cup \mathcal{X''}\) then \((\mathcal{X}) t \equiv t''\) is derivable.

1. Assume \(\alpha^* [\mathcal{X'}] t \equiv t'\) and \(\alpha^* [\mathcal{X''}] t' \equiv t''\)
2. \(\theta^*[t] = t'\) 1, truth for quantification Defn. 10
3. \(\theta^*[t'] = \theta^*[t']\) 2, assignment Defn. 8
4. \(\theta^*[t'] = \theta^*[t']\) transitivity in \(D_M\)
5. \(\theta^*[t'] = t''\) 4, assignment Defn. 8
6. \(\alpha^* [\mathcal{X}] t \equiv t''\) 5, truth for quantification Defn. 10.

**Substitutivity.** If \((\mathcal{X}_i) t_i \equiv t'_i\) is derivable and if \(\mathcal{X} = \cup \mathcal{X}_i\) for \(i = 0..n - 1\) then
1. for any $n$-ary function symbol $f$ in $\Sigma$,

$(\mathcal{X}) f(t_0, \ldots, t_{n-1}) \equiv f(t'_0, \ldots, t'_{n-1})$ is derivable:

(a) Assume $\alpha^*[(\mathcal{X}) t_i \equiv t'_i] = T$, $i = 0..n - 1$  

(b) $\theta^*[t_i] = \theta^*[t'_i]$, $i = 0..n - 1$  

(c) $\alpha[(\mathcal{X}) f(t_0, \ldots, t_{n-1}) \equiv f(t'_0, \ldots, t'_{n-1})] = T$  

(d) $\theta^*[f(t_0, \ldots, t_{n-1})] = \theta^*[f(t'_0, \ldots, t'_{n-1})]$  

(e) $\alpha^*[(\mathcal{X}) f(t_0, \ldots, t_{n-1}) \equiv f(t'_0, \ldots, t'_{n-1})]$  

2. for any $n$-ary predicate symbol $P$ in $\Sigma$, if $(\mathcal{X}) P(t_0, \ldots, t_{n-1})$ is derivable

then $(\mathcal{X}) P(t'_0, \ldots, t'_{n-1})$ is derivable:

(a) Assume $\alpha^*[(\mathcal{X}) t_i \equiv t'_i] = T$, $i = 0..n - 1$  

(b) $\theta^*[t_i] = \theta^*[t'_i]$, $i = 0..n - 1$  

(c) $\alpha[(\mathcal{X}) P(t_0, \ldots, t_{n-1})] = T$  

(d) $\langle \theta^*[t_0], \ldots, \theta^*[t_{n-1}] \rangle \in \alpha(P)$  

(e) $\langle \theta^*[t'_0], \ldots, \theta^*[t'_{n-1}] \rangle \in \alpha(P)$  

(f) $\alpha^*[(\mathcal{X}) P(t'_0, \ldots, t'_{n-1})] = T$  

Modus ponens. If $(\mathcal{X}) B_0, \ldots, B_m \rightarrow A$ is derivable and if $\theta : \mathcal{X} \rightarrow \Sigma(\mathcal{Y})$ is any substitution such that:

1. each $(\mathcal{Y}) \theta B_i$ is derivable for $i = 0..m - 1$, and

2. $(\mathcal{Y}) \theta(X_i)$ is derivable for each $X_i \in \mathcal{X}$,

then $(\mathcal{Y}) \theta A$ is derivable. 

Let $\theta_0 : \mathcal{Y} \rightarrow D_{\mathcal{M}}$ be an arbitrary sorted assignment.

1. Assume $\alpha^*[(\mathcal{Y}) \theta B_i] = T$, $i = 0..m - 1$  

2. $\theta_0^*[\theta B_i] = T$, $i = 0..m - 1$  

3. Assume $\alpha^*[(\mathcal{X}) B_0, \ldots, B_m \rightarrow A] = T$  

4. if $\theta_0^*[\theta B_i] = T$, $i = 0..m - 1$, then $\theta_0^*[\theta A] = T$  

5. $\theta_0^*[\theta A] = T$  

6. $\alpha^*[(\mathcal{Y}) \theta A] = T$

\[\square\]

\textbf{Theorem 6 (Completeness).} For any specification $S = (\Sigma, \Gamma)$, any formula $(\mathcal{X}) P(t_0, \ldots, t_{n-1})$ that is satisfied in all $S$-models is derivable from $S$ by Rules 1–5.

\textbf{Proof.} Suppose $(\mathcal{X}) P(t_0, \ldots, t_{n-1})$ is satisfied by all $S$-models, then it is satisfied by the specification term model $\mathcal{T}_S(\mathcal{X})$ defined above and $[(t_0, \ldots, t_{n-1})] \in \alpha[P]$. Then, by Definition 23 of $\alpha[P]$, $(\mathcal{X}) P(t_0, \ldots, t_{n-1})$ is derivable by Rules 1–5. \[\square\]

\textbf{Theorem 7 ($\mathcal{T}_S(\mathcal{X})$ is a free $S$-model).} Let $S = (\Sigma, \Gamma)$ be a specification, then $\mathcal{T}_S(\mathcal{X})$ is a free $S$-model over $\mathcal{X}$ in the class of all $S$-models.

\textbf{Proof.} Let $S = (\Sigma, \Gamma)$ be a specification, let $q : D_{\mathcal{T}_S(\mathcal{X})} \rightarrow D_{\mathcal{M}}$ be the quotient $\Sigma$-homomorphism and $\alpha$ be the interpretation function of Definitions 22 and 23. Let $\mathcal{M}$ be an $S$-model with interpretation function $\beta$ and let $\theta : \mathcal{X} \rightarrow D_{\mathcal{M}}$ be an arbitrary sorted assignment. It is required to
show that there is a unique $\Sigma$-homomorphism $\theta': D_{T_{\Sigma}(X)} \to D_{\mathcal{M}}$ such that $\theta = \theta' \circ q$. The proof proceeds in two steps. First the existence of $\theta'$ is shown and then its uniqueness.

$$
\begin{array}{c}
X \xrightarrow{q|X} D_{T_{\Sigma}(X)} \\
\theta \downarrow \theta' \downarrow \theta^*
\end{array}
$$

The existence of $\theta'$ is shown in two parts corresponding to the two-part definition of the quotient homomorphism $q$ in Definition 22 and its extension in Definition 23. By the soundness theorem, each derivable equation $(X) \ t_1 \equiv t_2$ is satisfiable in $\mathcal{M}$. That is, $\theta^*[t_1] = \theta^*[t_2]$, which means that $\equiv_{\Sigma} \subseteq \ker(\theta)$. Applying Theorem 2, the universal property of quotients, there is a unique homomorphism $\theta': D_{T_{\Sigma}(X)}/_{\equiv_{\Sigma}} \to M$ such that $\theta^* = \theta' \circ q$.

The second part of the existence proof deals with the homomorphism condition for predicates (Definition 12).

1. $\langle [t_0], \ldots, t_n \rangle \in \alpha[P]$ iff By Defn. 23 for representatives $t_i$ of $[t_i]$
2. $(X) P(t_0, \ldots, t_{n-1})$ is derivable from $S$ iff Soundness, Completeness theorems
3. $(X) P(t_0, \ldots, t_{n-1})$ is satisfied in all $S$-models
4. $\langle \theta^*[t_0], \ldots, \theta^*[t_n] \rangle \in \beta[P]$ 3, $\mathcal{M}$ is an $S$-model.

The uniqueness of $\theta'$ is shown by assuming there is another homomorphism $\theta'': D_{T_{\Sigma}(X)} \to D_{\mathcal{M}}$ such that $\theta = \theta'' \circ q$. Since $T_{\Sigma}(X)$ is a free model on $X$, $\theta$ is uniquely extended to $\theta''$ such that $\theta^* = \theta'' \circ q$. By Theorem 2 (the universal property of quotients) there is only one homomorphism satisfying this property and so $\theta' = \theta''$.

13 Signatures and type checking

An obvious difference between a semantically based method such as CM and other more syntactically based specification methods is that with the CM technique there are no operation sorts and any operation can be applied to any operand. Does this mean that we must abandon syntactic type checking? The answer is “no”, but with syntactic type checking we might not realize the full potential of the CM method.

For each sort symbol $s$ of a specification $S = (\Sigma, \Gamma)$ there is a set of expressions $t$ for which $s(t)$ is true in the initial model of $S$. Since this classification may not in general be decidable, we cannot expect to have an algorithm which checks the sorts of expressions. However, for any set of declaration assertions (without equations) in $\Gamma$ the syntactic classification is decidable. From any specification $S$ we can form another set $T$ of assertions such that:

1. each element of $T$ is true in the initial model of $S$ (i.e., it is true of the sorts specified);
2. the syntactic classification induced by $T$ is decidable.

We can say that the type checking with respect to $T$ is “partially correct”. The type checker can determine that an expression is of some sort, but there might be some expressions for which the type checker might not determine the least sort. For example, some value is always an integer even though the best that the type checker can do is classify it as of sort $\text{real}$. Fortunately, partial sort information is sufficient in many applications.

The CM technique allows the formation of expressions in an arbitrary fashion that is foreign to the more common strongly typed languages. For example, a queue can be added and integers can have a $\text{front}$ operation applied. A language interpreter does not have to provide a programmer with the full CM capability since the implementer could always select some set $T$ (as described above) and require that expressions in a program be “classifiable” according to $T$. For example,
given the OBJ2 (an OSA language) goal of executing specifications, it is probably appropriate to restrict the type checking to the syntactic form. Thus, OBJ2 can be considered a CM language in which an additional constraint has been imposed by the language designer. If the goals of a language include performing derivations about the properties of programs, it might be appropriate to adopt the general CM approach. The strength of a CM type discipline depends inversely on that of T since the formal CM system does not prescribe the strength but leaves the decision with the language designer (where it belongs). Milner’s ML language has inspired much of this work.

There is a need for some restrictions on programs and specifications in order to avoid consideration of “error” terms. This is the purpose of “normal” specifications in which each term that appears in the specification can be classified in the initial model. In combination with the rules of inference and the induction rule we are assured that unsorted terms cannot appear in a proof, since none of them appear in the specification and they cannot be introduced by the rules.

14 Related work: Order sorted techniques

A language implementer can, as described in the previous section, select a type mechanism that is more strict than the mechanism described for the Classified-Model concept. For example, a syntactic signature and a corresponding strong typing mechanism can be chosen because of language implementation constraints such as parsing considerations. An OSA language such as OBJ could be considered an implementation of CM because it restricts the language to a manageable subset that can be implemented.

The classified approach was intended for more than just execution of specifications. This broader purpose includes execution as well as deduction that can be used, for example, in ML-style type checking or for proof of specification properties.

The following subsections compare OSA and CM by first focusing on their similarities and then upon the characteristics that distinguish them.

14.1 Similarities between OSA and CM

Both OSA and CM are motivated by the desire to solve fundamental problems with MSA in the areas of overloaded operators and the treatment of errors. Both techniques find solutions in the introduction of subsorts, although they differ in the method of specifying subsorts. In this paper it is shown that OSA and CM can use similar rules of inference, although CM is formalized in a different setting with proofs similar to the published OSA formalization. In [5] it is shown that a version of OSA having a universal sort, similar to the CM approach, encompasses the published versions of OSA.

Parameterization issues are identical for OSA and CM, since both are liberal institutions [8]. This could be called “specification-in-the-large” because it relates the ways that components of a larger specification are glued together. The construction of basic specifications by either OSA or CM techniques could be called “specification-in-the-small”. The CM technique described in this paper supplies just “specification-in-the-small” syntax, whereas both types are included in OSA languages such as OBJ. Like the parameterization issue, both OSA and CM have the same module (ADTs with hidden states) concepts.

14.2 Differences between OSA and CM

OSA is an extension of MSA, whereas the classified method is not constrained to just a syntactic assignment of sorts to terms. That is, the classified type system, which includes the MSA and OSA syntactic typing as a special case, does not determine the sort of an expression strictly from the type of its subexpressions. Instead, sorts are determined according to assertions of the specification language.

Subsort declarations differ in OSA and CM according to the fundamental difference in their orientation. Subsort declarations of OSA are part of the syntactic signature, whereas they are
included in the assertions of the classified approach. The counterpart of OSA sort constraints is the CM use of sort predicates in the antecedent of a formula to constrain the use of some sort. For example, an OSA sort constraint is applied in a bounded-stack signature to limit the number of push terms. The CM counterpart uses an antecedent formula in a declaration assertion. Subsorts in the CM technique are usually declared inductively independently of the supersort, because this often results in simpler derivations of properties of the subsort. This is in contrast to the OSA practice of defining each sort in terms of its subsorts, although this can be derived from the inductive form.

A fundamental difference between models of OSA and CM is that a CM initial model contains terms that do not exist in the corresponding OSA initial model. These are “error” terms that correspond to unintended uses of the operations. In an OSA initial model some of these terms occur in “error supersorts” of the declared sorts, whereas in the CM technique any operation can be applied to any term(s). In the CM technique we can also declare error supersorts for the anticipated errors. The arbitrary application of operations to term(s) appears at first glance to present a problem. These terms, however, are not permitted in derivations from normal specification using the rules of inference and induction rule presented in this paper.

14.3 Unique aspects of order-sorted methods

The order-sorted methods have the following [3] characteristics:

Monotonicity condition. Suppose an operation “op” is “overloaded” by having both the declaration op : s₁, ..., sₙ → s and the declaration op : s₁', ..., sₙ' → s'. Suppose also the partial order includes sᵢ ≤ sᵢ' for i = 1, ..., n. Then for the specification to satisfy the monotonicity condition it must be the case that s ≤ s'.

Regular signature. Each term has a defined least sort.

The monotonicity condition ensures consistency between subsort declarations and operation declarations. That is, it ensures that the operation declarations do not imply a subsort relation contrary to the subsort declarations. In the CM system the monotonicity condition is unnecessary because subsort relations are inferred from the term sorts that are declared by the Horn assertions, so there can be no conflict in declarations. In the CM system the best we can do with regard to regularity is to prove some sort for each term, but it might not be the least sort.

15 Conclusion

The Classified-Model (Horn with equality) specification technique has a defined syntax and semantics that has been formalized as an alternative to Many-Sorted and Order-Sorted techniques for the specification of abstract data types (ADTs) and modules (ADTs with hidden state sorts). The main contribution of this result is that the Order-Sorted Model (OSM) rules of inference can be used within the classified technique and that the unclassified, or error terms, cause no difficulties in practice.

The basic classified method may be called “specification in the small” because it deals with the construction of individual assertions. “Specification in the large” is addressed by existing parameterization techniques that can be used to glue component specifications into larger units. Horn with equality is a sufficiently restrictive language to qualify as a “liberal institution” and therefore inherit many existing parameterization techniques.

References


