Computability in Specification

RAYMOND TURNER, Department of Computer Science, University of Essex, Colchester, Essex CO4 3SQ, UK. E-mail: turner@essex.ac.uk

Abstract

In reference (Foundation of specification. Journal of Logic and Computation, 15, 951–974, 2005), the author introduces a core specification theory (CST) in order to provide a logical framework for the design and exploration of specification languages. In this article, we formulate two highly expressive extensions of CST. The first (CST_U) is CST + a universe of types and the second (CST_US) permits specifications themselves to be data items. Finally, we shall explore their metamathematical properties and, in particular, provide an interpretation into first-order arithmetic.

Keywords: Data, types, specification, Σ-definable, conservative extensions, polymorphism.

1 The role of computability

We have referred to axiomatizations of logic-based specification languages (SLs) (e.g. 8, 9, 16–18, 20, 38, 21, 27, 33, 7, 14, 22) as specification theories (STs) [35]. Our objective in this second article is to formulate and investigate STs in which computability considerations play a significant role. This influence is to apply both to the design of the SLs and to the nature and form of specifications. To motivate matters, we make some preliminary observations.

Not all mathematical objects can serve as data items, i.e. the kinds of things that get manipulated in programming languages. Examples of the latter include numbers, lists, trees, finite sets, pairs, objects and classes. Indeed, programming languages equip these items with basic relations and functions, including equality, which constitute an implementable or computable data type (CDT) [4]. In contrast, many current specification languages are not based upon CDTs. In particular, many such languages [1, 8, 26, 24, 11, 32] have some rendering of higher-order logic (HOL) or even Zermelo–Fraenkel set theory (ZF) at their core. But on the assumption that \( N \), the set of natural numbers, is a type and that we can form the type of all subsets of a given type, \( P(N) \) is a type. However, these infinite sets given in extension cannot be turned into a CDT—equality for them is not even semi-decidable. Consequently, extensional HOL with the type of natural numbers at its base is not, in our sense, a theory of data. Parsimony and computational good sense suggest that one should develop and explore theories that are.

Indeed, we claim that the majority of current specification languages, although adequate for computational practice, are not faithful to that practice, where faithfulness and adequacy [13] are specified in [5] as follows:

- A formal theory \( T \) is an adequate formalization of an informal body of mathematics \( M \) if every concept, argument and result of \( M \) can be represented by a basic (or defined) concept, proof and theorem, respectively, of \( T \).
- \( T \) is faithful to \( M \) if very basic concept of \( T \) corresponds to, or is implicit in, the assumptions and reasoning followed in \( M \), i.e. \( T \) does not go beyond \( M \) conceptually or in principle.

\[1\] In this regard, VDM, [24], which admits function space types, is to be contrasted to VDM-SL, [20].
For example, let $M$ be the body of number theory that one finds in books on elementary number theory and let $T$ be the formal theory known as Peano arithmetic formulated with function symbols for each primitive recursive function. Then $T$ is adequate and faithful to $M$. In contrast, if $T$ is ZF, then, while it is clearly adequate, it is certainly not a faithful formalization: standard representations of the numbers have too many extraneous properties. Similarly, if $M$ is classical analysis and $T$ is ZF then, while $T$ is adequate, it is not faithful; much weaker theories are adequate, e.g. the theory $Z_2$, which goes under the name classical analysis.

Do current STs adequately and faithfully formalize the computational practice of everyday specification? Here we are not talking about some contrived examples in which these languages are used to define themselves; obviously, ZF will be needed to provide a definition of itself. Rather, we are concerned with what gets specified in the actual applications of specification languages such as Z and The Vienna Development Method (VDM) to program specification. After all, our primary goal is to specify programs operating over CDTs, i.e. we aim to specify what programming languages can encode.

Despite their somewhat incomplete axiomatic specifications, it seems clear that they are adequate for this practice; indeed, they are the vehicles of it. However, their faithfulness is suspect; the expressive power of these theories goes way beyond what is needed in practice. In particular, as we have already observed, most are not theories of data. But surely program specifications should operate over the things that programs operate over. Otherwise, we must provide powerful arguments to the contrary. And I have yet to see any. It would seem to me very counter-intuitive if $Z_2$ yielded a faithful formalization of analysis, while the whole of HOL (or even the whole of ZF) was necessary for the specification of computer programs. To be clear, we are not suggesting that SL should only be able to specify computable relations and functions; that is a more complicated issue that we shall address later. We are only arguing that, to be faithful to the practice of specification, they have to be theories of what we write program specifications about, i.e. they must be theories of data.

But even if we concede this general point and allow specification theories that are not theories of data, these theories are still not faithful. For one thing, we still have to represent our data types in these theories. Unfortunately, the formalization of CDTs in HOL/ZF is not faithful; it is no more faithful than the formalization of informal number theory in ZF, i.e. the argument in [6] applies generally to all data types. Secondly, even though the logics of many of these theories contains at least the set theory of HOL, the everyday use of set theory in specification in VDM or Z is not that of HOL but only that of CST. How could it be otherwise? Well, I suppose that we could argue that we might wish to write specifications where the data objects are functions, i.e. the specification of functional programs. But the last phrase gives the game away: it is not set theoretic functions that are the data items but the functional programs themselves.

On the other hand, we are under an obligation to demonstrate that theories of data are adequate. This was started in [35] and is continued here. However, although CST is a theory

---

2 [6] tells the story of two little boys who are each taught a different representation of the natural numbers. Eventually, they get into a dispute about which is the right one, e.g. is $2 \in 3$? But of course there is not a right one; there is no obvious way of choosing between them.

3 Current specification languages are rarely precisely and completely formulated as axiomatic theories. However, we do have some fragments, e.g. [8, 20].
of data that faithfully reflect much computational practice, it is rather limited in its expressive power.\(^4\) In this article, we formulate two highly expressive extensions of CST that are still theories of data. The first (CST\(_U\)) can be roughly characterized as CST + a universe of types. It is actually an extension of the theory CST\(_T\) [35], i.e. the logic and the type inference system are intermingled. This formulation, although over the top for CST itself, more elegantly supports more expressive theories. The second (CST\(_{US}\)) permits specifications themselves to be data items. This facilitates the representation of, what are normally regarded as, higher-order notions within a theory of data. These two extensions are not only pragmatically significant but they also offer technical challenges: how to design an ST that has such expressive power and yet be a theory of data.

2 The theory CST\(_U\)

We employ a many sorted system of natural deduction, the conclusions of which are judgements of one of the following four forms.\(^5\)

\[ t : T \]
\[ T \text{ type} \]
\[ \phi \text{ prop} \]
\[ \phi \]

The first asserts that \( t \) is an object term of type \( T \), the second that \( T \) is a type and the third that \( \phi \) is a proposition. We shall refer to these three as type inference judgements. The last, the logical one, asserts that \( \phi \) is true. These judgements are made relative to a context \( \Gamma \)—a finite sequence of judgements of one of the two forms: \( x : T \) or \( \phi \), i.e. a declaration that a variable has a given type or the assertion that a proposition is true. Thus, sequents in the theory have the shape:

\[ \Gamma \vdash \Theta \]

where we employ \( \Theta, \Theta' \) as meta-variables for our four judgement forms and uppercase Greek letters for contexts.\(^6\)

These judgements are formed from a background syntax of terms and, well formed formulae (wff). Terms are built from variables \( (v_0, v_1, v_2, v_3, \ldots) \), certain constants \( (0, \emptyset, N, U) \) and operators \( (+, \varnothing, \pi_1, \pi_2, [, ] , \text{Set}, \Sigma) \). The wff are constructed from the object terms via

---

\(^4\) The theory behind VDM-SL is also such a theory and it is similarly limited in expressive power. However, the full version of VDM is not.

\(^5\) Although the constructive paradigm in specification is quite different to the ones considered here, the formulation of our theory has been influenced in its style by the constructive type theories of Martin-Löf [5, 23].

\(^6\) Note that these are not sets. Dependency requires them to be sequences since a later type assignment being in a sequence is legitimate depends upon whether the type term is a type, and this may depend upon earlier members of the context.
certain operators (\(\in\), \(=\), \(<\)), the logical connectives (\(\Omega\), \(\wedge\), \(\vee\), \(\to\), \(\neg\)) and the numerical, set and type quantifiers. As meta-variables, we shall use \(x, y, z, u, v, w\) for variables; \(t, s, T, S\) for object terms and lowercase Greek letters for wff. More exactly, we adopt the following Backus Naur Form (BNF) syntax for these syntactic classes.

\[
\begin{align*}
t \ ::= & \ x | 0 | t^+ | (t, t) | \pi_1(t) | \pi_2(t) | \emptyset | t \otimes t | N | U | \Sigma x : t \cdot t | \text{Set}(t) | S(t) \\
\phi \ ::= & \ \Omega | t | t \in t | t \leq t | \neg \phi | \phi \land \phi | \phi \lor \phi | \phi \rightarrow \phi | \forall x \in t \cdot \phi | \exists x \in t \cdot \phi
\end{align*}
\]

where \(x \in t\) is one of \(x < t\) (numerical ordering), \(x : t\) (type membership) or \(x \in s\) (set membership). Here, \(+\) is the successor, \(\emptyset\) the empty set, \(\otimes\) the operation for adding an element to a set, \((\) and \(\pi\) for pairing. \(N\) is the type of natural numbers, \(U\) is a universe of types, \(\text{Set}\) forms the type of sets and \(\Sigma\) forms dependent sums.\(^7\)

However, from the perspective of the type system, this syntax is too permissive, since it takes no account of the types of variables. Hence, a subset of the rules act as a type inference system [2, 34, 35].

In stating the rules, we shall only indicate changes in the context in which there is a change from premises to conclusion. We begin with the structural ones, i.e. assumption, thinning and substitution.

\[
\begin{align*}
A_1 & \quad \Gamma \vdash T \text{ type} \\
A_2 & \quad \Gamma \vdash \phi \text{ prop} \\
W_1 & \quad \Gamma, x : T \vdash \emptyset \\
W_2 & \quad \Gamma, x : T, \Delta \vdash \emptyset \\
\text{Sub} & \quad \Gamma, x : T, \Delta \vdash \emptyset \\
\text{Sub} & \quad \Gamma \vdash t : T \\
\end{align*}
\]

where in \(A_1\) and \(W_1\), \(x\) is not declared in \(\Gamma, \Delta\). Although some instances of these rules are derivable from the others, for convenience, we take them all as basic. Note that we do not have an exchange rule because the contexts are dependent. For example, the use of type terms in a context may depend upon previous type declarations.

We next provide the formation, introduction and elimination rules for equality, the propositional connectives and the quantifiers. The equality formation rule insists that equality forms a proposition when the terms flanking it have the same type. The other two are the standard introduction and elimination rules for equality.

\[
\begin{align*}
E_1 & \quad t : T \quad s : T \\
E_2 & \quad t : T \\
E_3 & \quad t = s \quad \phi[t/x] \\
\end{align*}
\]

\(^7\)These types are those of CST plus a universe of types that replaces type quantification. This permits the generalization of standard products to dependent sums.
The formation rules for the propositional connectives \((L_1, L_5, L_9, L_{12}, L_{15})\) mimic the standard syntax, and the introduction and elimination rules are those of classical logic.

\[
\begin{align*}
L_1 & \quad \phi \text{ prop} \quad \psi \text{ prop} \\
& \quad \frac{}{\phi \land \psi \text{ prop}} \\
L_5 & \quad \phi \text{ prop} \quad \psi \text{ prop} \\
& \quad \frac{}{\phi \lor \psi \text{ prop}} \\
L_7 & \quad \phi \quad \psi \text{ prop} \\
& \quad \frac{}{\phi \lor \psi} \\
L_9 & \quad \phi \text{ prop} \quad \psi \text{ prop} \\
& \quad \frac{}{\phi \to \psi \text{ prop}} \\
L_{11} & \quad \phi \quad \psi \text{ prop} \\
& \quad \frac{}{\phi \to \psi} \\
L_{13} & \quad \Gamma, \phi \vdash \Omega \\
& \quad \frac{}{\Gamma \vdash \phi} \\
L_{15} & \quad \Omega \text{ prop} \\
& \quad \frac{}{\Gamma \vdash \phi}
\end{align*}
\]

The type quantifier rules are given as follows. We assume the standard side conditions, i.e. in \(L_{20}\), \(x\) must not be free in any proposition in \(\Gamma\) and \(\eta\), and in \(L_{22}\), \(x\) must not be free in any proposition in \(\Gamma\).

\[
\begin{align*}
L_{18} & \quad \Gamma, x : T \vdash \phi \text{ prop} \\
& \quad \frac{}{\Gamma \vdash \exists x : T \cdot \phi \text{ prop}} \\
L_{20} & \quad \exists x : T \cdot \phi \\
& \quad \frac{\Gamma \vdash \eta}{\Gamma \vdash \exists x : T, \phi \vdash \eta} \\
L_{21} & \quad \Gamma, x : T \vdash \phi \text{ prop} \\
& \quad \frac{}{\Gamma \vdash \forall x : T \cdot \phi \text{ prop}} \\
L_{22} & \quad \Gamma, x : T \vdash \phi \\
& \quad \frac{}{\Gamma \vdash \forall x : T \cdot \phi} \\
L_{23} & \quad \exists x : T \cdot \phi \quad t : T \\
& \quad \frac{\phi[t/x]}{\Gamma \vdash \forall x : T \cdot \phi \vdash t : T}
\end{align*}
\]

This completes the logical rules. We shall give the bounded quantifier rules with their corresponding types. Indeed, we now turn to the content of these individual type constructors, beginning with the numbers.

The first rule asserts that \(N\) is a type and the next five are those of Peano arithmetic (PA). The last few provide the formation, introduction and elimination rules for the ordering relation.

\[
\begin{align*}
N_0 & \quad N \text{ type} \\
N_1 & \quad 0 : N \\
N_2 & \quad a : N \\
N_3 & \quad \frac{\Gamma \vdash \phi[0]}{\Gamma, x : N \vdash \phi[x] \to \phi[x^+]}
\end{align*}
\]

This completes the logical rules. We shall give the bounded quantifier rules with their corresponding types. Indeed, we now turn to the content of these individual type constructors, beginning with the numbers.

The first rule asserts that \(N\) is a type and the next five are those of Peano arithmetic (PA). The last few provide the formation, introduction and elimination rules for the ordering relation.

\[
\begin{align*}
N_0 & \quad N \text{ type} \\
N_1 & \quad 0 : N \\
N_2 & \quad a : N \\
N_3 & \quad \frac{\Gamma \vdash \phi[0]}{\Gamma, x : N \vdash \phi[x] \to \phi[x^+]}
\end{align*}
\]

This completes the logical rules. We shall give the bounded quantifier rules with their corresponding types. Indeed, we now turn to the content of these individual type constructors, beginning with the numbers.

The first rule asserts that \(N\) is a type and the next five are those of Peano arithmetic (PA). The last few provide the formation, introduction and elimination rules for the ordering relation.
The next batch govern the numerical ordering relation.

\[
\begin{align*}
N_6 & : a : N \quad b : N \quad a < b \text{ prop} \\
N_7 & : a : N \quad a < a^+ \\
N_8 & : a : N \quad a < b^+ \quad a \neq b \\
\end{align*}
\]

Finally, the rules for the numerical quantifiers are given in terms of type quantification, as follows.

\[
\begin{align*}
N_{11} & : \Gamma, x : N \vdash \phi \text{ prop} \quad \Gamma \vdash s : N \\
N_{12} & : \Gamma \vdash \forall x < s \cdot \phi \text{ prop} \\
N_{13} & : \Gamma, x : N \vdash \phi \text{ prop} \quad \Gamma \vdash \exists x < s \cdot \phi \text{ prop} \\
N_{14} & : \Gamma \vdash s : N \quad \Gamma, x : N \vdash \phi \text{ prop} \\
\end{align*}
\]

The dependent sum generalizes the Cartesian product constructor of CST to allow for dependency. Where \( x \) is not free in \( B \), we obtain the standard Cartesian product, \( A \otimes B \). In particular, we shall write \( A^n \) for the Cartesian product of \( n \)-copies of \( A \). We shall often write \( \pi_i(x) \) as \( x_i \).

\[
\begin{align*}
P_0 & : x : A \vdash B \text{ type} \\
P_1 & : a : A \quad b : B[a/x] \\
P_2 & : a : \Sigma x : A \cdot B \\
P_3 & : a : \Sigma x : A \cdot B \\
P_4 & : (a, b) : \Sigma x : A \cdot B \\
P_5 & : a : \Sigma x : A \cdot B \\
\end{align*}
\]

The formation rule for sets, \( S_0 \), insists that the types are closed under set formation. The rest of the rules are essentially those of CST. Rules \( S_1-S_3 \), present set types as inductive. \( S_4 \) and \( S_5 \) govern the special equality axioms.
The last five determine set membership:

\[ S_6 \quad a : T \quad b : \text{Set}(T) \quad a \in b \quad \text{prop} \]
\[ S_7 \quad a : T \quad b : \text{Set}(T) \quad a \in a \oplus b \]
\[ S_8 \quad a = c \quad b : T \quad c : \text{Set}(T) \quad a \in b \oplus c \]
\[ S_9 \quad a \in \emptyset \quad \Omega \]
\[ S_{10} \quad a \in b \oplus c \quad a \neq b \quad a \in c \]

The rules for the set quantifiers follow those for the numerical ones, i.e. they take the following shape:

\[ S_{11} \quad \Gamma, x : T \vdash \phi \quad \text{prop} \quad \Gamma \vdash s : \text{Set}(T) \quad \Gamma \vdash \forall x \in s : \phi \quad \text{prop} \]
\[ S_{12} \quad \Gamma \vdash \forall x \in s : \phi \iff \forall x : T \cdot x \in s \rightarrow \phi \]
\[ S_{13} \quad \Gamma, x : T \vdash \phi \quad \text{prop} \quad \Gamma \vdash s : \text{Set}(T) \quad \Gamma \vdash \exists x < s : \phi \quad \text{prop} \]
\[ S_{14} \quad \Gamma \vdash \exists x \in s : \phi \iff \exists x : T \cdot x \in s \land \phi \]

Finally, to support quantification over types, and a form of generic or polymorphic specification, we introduce a type \( U \), the type of types. This is the substance of the following three rules:

\[ U_1 \quad U \quad \text{type} \]
\[ T \quad \text{type} \]
\[ T : U \quad \text{type} \]

These generalize the type quantification of CST. Note that the judgement \( T \text{ type} \) is actually technically redundant, but it is conceptually helpful. The types are types of data items—including the type \( U \). We shall justify this later, but for now one should note there is no axiom of extensionality for types.

This completes the statement of the theory CST\(_U\). We now document some of its elementary consequences. The last part of the following demonstrates the extensional nature of sets. The essence of the proof can be found in [35].

**Proposition 1**

For all \( A \) of type \( U \):

(1) \( \forall y : \text{Set}(A) \cdot \forall x : A \cdot x \oplus y \neq \emptyset \)

(2) \( \forall y : \text{Set}(A) \cdot \forall x \in y : x \oplus y = y \)

(3) \( \forall z : \text{Set}(A) \cdot \forall x \in z : \exists y : \text{Set}(A) \cdot x \neq y \land z = x \oplus y \)

(4) \( \forall x : \text{Set}(A) \cdot \forall y : \text{Set}(A) \cdot ((\forall z \in x \cdot z \in y) \land (\forall z \in y \cdot z \in x)) \rightarrow x = y \)

The following principle of collection for sets we shall need later:

**Proposition 2** (Collection)

\( \forall u : U \cdot \forall v : U \cdot \forall x : \text{Set}(u) \cdot (\forall y \in x : \exists z : v \cdot \psi[y, z]) \rightarrow (\exists w : \text{Set}(v) : \forall y \in x \cdot \exists z \in w \cdot \psi[y, z]) \)

**Proof.** Let \( A : U \) and \( B : U \). Let \( x : \text{Set}(A) \). We use set induction with the induction formula:

\[ \phi[x] = \forall y \in x : \exists z : B \cdot \psi[y, z] \rightarrow \exists w : \text{Set}(B) : \forall y \in x : \exists z \in w \cdot \psi[y, z] \]
If $x$ is the empty set, it is immediate. Assume that $x = x' \otimes v$ and assume the result for $v$. Assume that $\forall y \in x' \otimes v \cdot \exists z : B \cdot \psi[y, z]$. By this and induction, $\exists w' : Set(B) \cdot \forall y \in v \cdot \exists z \in w' \cdot \psi[y, z]$. Also, since $x' \in x$, we have $\exists v' : B \cdot \psi[x', v']$. The required set is $v' \otimes w'$.

We next abstract and study the type inference system [34, 2]. Since the modifications to the original system of CST are few, we shall be brief. We shall call contexts that contain only type assignments (i.e. of the form $x : T$) as declaration contexts. We shall use $c, c', d, d'$, etc. as meta-variables for these contexts and $c_\Gamma$ for that part of the context $\Gamma$ that consists of just its type assignments.

**Definition 3**

The subtheory $T$ is that theory whose sequents are all of the form

$$c \vdash \Theta$$

where $\Theta$ is a type inference judgement.

We first establish this theory is independent of the main one.

**Proposition 4 (Independence)**

If $\Gamma \vdash \Theta$, where $\Theta$ is a type inference judgement, then $c_\Gamma \vdash_T \Theta$.

**Proof.** By induction on the rules with type inference conclusions. Observation of these demonstrates that they only require declaration contexts and type inference premises.

The following provides the basis for a type-checking algorithm:

**Proposition 5 (Type-checking)**

In $T$ we have:

1. $c \vdash t = s \text{ prop iff } c \vdash t : T$ and $c \vdash s : T$, for some type $T$
2. $c \vdash t \in s \text{ prop iff } c \vdash t : T$ and $c \vdash s : \text{Set}(T)$, for some type $T$
3. $c \vdash t < s \text{ prop iff } c \vdash t : N$ and $c \vdash s : N$
4. $c \vdash \phi \circ \psi \text{ prop iff } c \vdash \phi \text{ prop and } c \vdash \psi \text{ prop, } \circ = \lor, \land, \rightarrow$
5. $c \vdash Q \text{set} \cdot \phi \text{ prop iff } c, x : \text{set} \vdash \phi \text{ prop for } Q = \forall, \exists$ and $\varepsilon = \langle, \in :$
6. $c \vdash s \otimes t : \text{Set}(T)$ iff $c \vdash s : T$ and $c \vdash t : \text{Set}(T)$
7. $c \vdash (a, b) : \Sigma x : A \cdot B$ iff $c \vdash a : A$ and $c \vdash b : B[a/x]$  
8. $c \vdash \pi_1(a) : A$ and $c \vdash \pi_2(a) : B[a/x]$ iff $a : \Sigma x : A \cdot B$
9. $c \vdash t : U$ iff $c \vdash t \text{ type}$
10. $c \vdash \text{Set}(A)$ type iff $c \vdash A$ type
11. $c \vdash \Sigma x : A \cdot B$ type iff $c \vdash A$ type and $c, x : A \vdash B$ type

**Proof.** The proofs of results in the direction from right to left follow immediately from the rules. For the other direction, we use induction on the structure of derivations. If the conclusion follows from the introduction rules for the connective, the result is immediate. If the conclusion is the result of a structural rule, the result follows from using the structural rule itself. For example, suppose the last step in the derivation is the following instance of an application of $W_1$.

$$c \vdash T \text{ type}, c \vdash \phi \land \psi \text{ prop}$$

$$c, x : T \vdash \phi \land \psi \text{ prop}$$
Consider the premises. By induction, we may suppose that \( c \vdash \phi \) \( \text{prop} \) and \( c \vdash \psi \) \( \text{prop} \).

By induction, \( c, x : T \vdash \phi \) \( \text{prop} \) and \( c, x : T \vdash \psi \) \( \text{prop} \).

The rules of \( \text{CST}_U \) ensure that only propositions are provable. We shall call this the \textit{coherence} property of the logic.

**Theorem 6 (Coherence)**

(1) If \( \Gamma \vdash \phi \) then \( c_\Gamma \vdash \phi \) \( \text{prop} \)

(2) If \( c \vdash t : T \) then \( c \vdash T \) \( \text{type} \)

(3) If \( \Gamma, x : T, \Gamma' \vdash \Theta \) then \( c_\Gamma \vdash T \) \( \text{type} \)

(4) If \( \Gamma, \phi, \Gamma' \vdash \Theta \) then \( c_\Gamma \vdash \phi \) \( \text{prop} \)

**Proof.** The proof is by induction on the structure of derivations. All the cases are routine. We illustrate part 1 with the following case of the existential quantification introduction.

\[
L_{19} \quad \frac{\Gamma \vdash \phi[t/x]}{\Gamma \vdash \exists x : T \cdot \phi}
\]

By induction,

\[
c_\Gamma, x : T \vdash \phi \text{ prop}
\]

Now use the existential formation rule. For part 2, we illustrate with \( P_1 \). Here we apply the induction hypothesis to the assumptions, use the substitution proposition to get the variables in the context and finish off with the formation rule for dependent products. Parts 3 and 4 are equally easy to check.

### 3 \( \Sigma \)–specifications

In [11], we introduced a notion of relation specification based upon the Z schema notation. In this section, we give it a computational makeover. Later we shall discuss its significance and limitations and how it fits with current specification regimes.

Because ours is a theory of computable data types, it makes immediate sense to talk about computing with them. In particular, it makes sense to generalize the standard notion of \( \Sigma \)-definability to the objects of \( \text{CST}_U \). In formulating our notion, we have been influenced by [25, 3, 4, 12, 30] and unpublished work by Hodges. To facilitate matters, we require two subclasses of \( \text{wff} \) that bring traditional definability notions into play.

**Definition 7**

The \( \Sigma \) \( \text{wff} \) are singled out via the following BNF syntax:

\[
t ::= x | 0 | t^+ | (t, t) | \pi_1(t) | \pi_2(t) | \emptyset | t \otimes t | N | U \Sigma x : t \cdot t | \text{Set}(t)
\]

\[
\alpha ::= \Omega | t = t | t \in t | t \leq t
\]

\[
\sigma ::= \alpha | \neg \sigma | \sigma \wedge \sigma | \sigma \vee \sigma | \forall x : t \cdot \sigma | \exists x : t \cdot \sigma
\]

where here \( \varepsilon \) is \( < \) or \( \in \).

A \( \Delta \) \( \text{wff} \) is one that is \( \Sigma \) and whose negation is, within the theory \( \text{CST}_U \), provably equivalent to a \( \Sigma \) one. The \( \Delta_0 \) \( \text{wff} \) are given by

\[
\delta ::= \alpha | \neg \delta | \delta \wedge \delta | \delta \vee \delta | \forall x : t \cdot \delta | \exists x : t \cdot \delta
\]

where \( \varepsilon \) is \( < \) or \( \in \).
To further explore some of the implications of our computational strictures, we introduce a sub-theory.

**Definition 8**
PRCST is that sub-theory in which the induction principles for numbers and sets are restricted to $\Sigma$ wff i.e., in $N_3$ and $S_3$, $\phi$ must be $\Sigma$.

We may now make precise our notion of computable schema specification.

**Definition 9**
Let $\phi$ be $\Sigma$ and

$$x : T \vdash \phi \text{ prop} \quad (R_0)$$

Then a Schema Specification has the form

$$R[x : T | \phi] \quad (R)$$

$R$ introduces a new relation symbol into the language that is constrained by the following type inference rule

$$a : T \quad \frac{}{R(a) \text{ prop}} \quad (R_1)$$

and is taken to satisfy:

$$x : T \vdash R(x) \leftrightarrow \phi[x] \quad (R_2)$$

Many place relations can be introduced using products; we shall write

$$R \triangleq [x_1 : T_1, \ldots, x_k : T_k | \phi[x_1, \ldots, x_k]]$$

for

$$R \triangleq [x : \Sigma x_1 : T_1 \cdot \ldots \cdot \Sigma x_{k-1} : T_{k-1} \cdot T_k | \phi[x_1, \ldots, x_k]]$$

We shall call $x_1 : T_1, \ldots, x_k : T_k$ the declaration of the schema and $\phi$ its predicate. Following Z, we shall also write schema specifications in the following more graphic form where, in this presentation, we may mark conjuncts with a new line and mark inputs to an operation with ? and outputs with !. However, as explained in [35], our interpretation of the schema is not that of Z.
Since the universe is the only addition to CST, we shall illustrate its use on a couple of examples. The first illustrates how it supports explicit polymorphism: the new relation \( \subseteq \) takes types, i.e. variables declared to be in the universe of types, as arguments.

**Example 10 (Subset)**

\[
\begin{array}{c}
\subseteq \\
\text{u : } U, \ x : \text{Set}(u), \ y : \text{Set}(u) \\
\forall w \in x \cdot w \in y \\
\forall w \in y \cdot w \in x
\end{array}
\]

This is \( \Sigma \) and clearly satisfies \( R_0 \). Moreover \( R_2 \) unpacks to:

\[
u : U, \ x : \text{Set}(u), \ y : \text{Set}(u) \vdash x \subseteq u \iff (\forall w \in x \cdot w \in y) \land (\forall w \in y \cdot w \in x)
\]

where we have used infix notation and written the type variable argument as a subscript. We shall often suppress the variables of type \( U \) in the declaration and recover them from their occurrence in the text. We facilitate this by using upper case variables to distinguish them.

\[
\begin{array}{c}
\subseteq \\
x : \text{Set}(X), \ y : \text{Set}(X) \\
\forall w \in x \cdot w \in y \\
\forall w \in y \cdot w \in x
\end{array}
\]

This now looks much like the generic specifications of [35].

Our next example illustrates the use of the universe to deliver new type constructors. It specifies an operator that, given two input types, returns the type of set theoretic relations over these inputs.

**Example 11 (Set theoretic relations)**

\[
\text{Relset} \triangleq [u? : U, v! : U, w! : U \mid w = \text{Set}(u \otimes v)]
\]

There are many more examples in [35], many of which are \( \Sigma \) and all of which can be reformulated as \( \Sigma \)-specifications. This should not be a surprise.

In [35], we insisted that all specifications should generate conservative extensions. The proof of the following needs to be slightly modified to cope with the
new formulation. Let CST\(^R\)\(_U\) be the theory CST\(_U\) with the rules \(R_1\) and \(R_2\) added. We take \(R(t)\) to be \(\Sigma\) in CST\(^R\)\(_U\).

**Lemma 12 (Soundness)**

There is a translation * from the syntax of CST\(^R\)\(_U\) to that of CST\(_U\) such that

1. If \(\Gamma \vdash_{\text{CST}\(_U\)} \Theta\) then \(\Gamma^* \vdash_{\text{CST}\(_U\)} \Theta^*\)
2. If \(c \vdash_{\text{CST}\(^R\)\(_U\)} \Theta\) then, where \(\Theta\) does not contain \(R\), \(\Theta^* = \Theta\)
3. If \(\psi\) is \(\Sigma\) in the extended theory, then its translation is \(\Sigma\) in the original.

where \(\Gamma^*\) is the translated context and \(\Theta^*\) is the translated judgement.

**Proof.** Let * be the translation between the two raw BNF definitions obtained by replacing, in the expressions of CST\(^R\)\(_U\), every occurrence of \(R(t)\) by \(\phi(t)\). Part 1 follows by induction on the derivations in CST\(^R\)\(_U\). All the rules of \(\text{Th}\) are easy to verify. This leaves us to check the new axiom, and this is immediate. The proofs of 2 and 3 are by inspection.

It follows that CST\(^R\)\(_U\) is a conservative extension of CST\(_U\), i.e.

**Theorem 13 (Conservative extension for relations)**

Suppose that \(\Gamma, \Theta\) are expressed in the BNF syntax of CST\(_U\). Then

\[
\Gamma \vdash_{\text{CST}\(_U\)} \Theta \text{ implies } \Gamma \vdash_{\text{CST}\(_U\)} \Theta
\]

We now review the addition of function symbols [19, 10]. Their introduction is somewhat more delicate [3, 36]. To illustrate matters we employ the binary case.\(^8\)

**Definition 14**

We shall say that a schema \(R \triangleq [x : I, y : O | \phi]\) is total if

\[
\forall x : I \cdot \exists y : O \cdot \phi[x, y]
\]

and functional if

\[
\forall x : I \cdot \exists! y : O \cdot \phi[x, y]
\]

where \(\exists! y\) indicates precisely one.

**Definition 15**

Let \(R \triangleq [x : I, y : O | \phi]\) functional. We may then introduce a new function symbol, \(F\), into the language governed by the grammar rule

\[
\frac{a : I}{F(a) : O[a]} \quad (F_1)
\]

and that satisfies

\[
\forall x : I \cdot \phi[x, F(x)] \quad (F_2)
\]

We first show that the theory supports the specification of some fundamental set-theoretic notions.

\(^8\) We shall not discuss functions with pre-conditions here, but see [35] for a treatment in the original version of CST. This may be adapted to the present setting by ensuring that the type system takes pre-conditions into account.
Example 16 (Union)

\[
\begin{align*}
\cup \quad u : \text{Set}(\text{Set}(X)) \quad v : \text{Set}(X) \\
(\forall z \in v \cdot \exists w \in u \cdot z \in w) \\
\forall w \in u \cdot \forall z \in w \cdot z \in v
\end{align*}
\]

Example 17 (Power set)

\[
\begin{align*}
P \quad x : \text{Set}(X) \quad y : \text{Set}(\text{Set}(X)) \\
\emptyset \in y \land (\forall z \in x \cdot \forall w \in y \cdot z \uplus w \in y) \\
\forall z \in y \cdot z \subseteq x
\end{align*}
\]

Proposition 18 (Union and power set)

The specifications of generalized union and power set are functional. Indeed, this can be established in PRCST.

Proof. They are easy to establish. For example, for the first, for totality, we use the \( \Sigma \) induction formulae

\[
\phi[u] = (\forall z \in v \cdot \exists w \in u \cdot z \in w) \land \forall w \in u \cdot \forall z \in w \cdot z \in v
\]

Functionality then follows by extensionality.

The following example is schematic with respect to the included proposition; it is a specification of separation for sets.

Proposition 19 (Separation)

Suppose \( u : U, z : u \vdash \psi \text{ prop} \) where \( \psi \) is \( \Delta \) i.e., \( \eta \leftrightarrow \neg \psi \) where \( \eta \) is \( \Sigma \). Specify

\[
\begin{align*}
\text{Sep}^\psi \\
u : U, y : \text{Set}(u) \quad z : \text{Set}(u) \\
(\forall x \in z \cdot x \in y \land \psi[x]) \land (\forall x \in y \cdot \eta[x] \lor x \in z)
\end{align*}
\]

This is functional and can be established in the restricted theory.
PROOF. Given the extensional nature of sets, functionality is immediate from the predicates. For totality, we use set induction with the \( \Sigma \) proposition:

\[
\sigma[y] = \exists z : \text{Set}(u) \cdot (\forall x \in z \cdot x \in y \land \psi[x]) \land (\forall x \in y \cdot \eta[x] \lor x \in z).
\]

We demonstrate that such functional additions are conservative. For \( \text{CST}_U \), the proof is somewhat more delicate. Suppose that we can prove \( F_0 \) in \( \text{CST}_U \). Let \( \text{CST}_U^F \) be \( \text{CST}_U \) with a new function symbol \( F \), governed by the rules \( F_1 \) and \( F_2 \), added. In the definition of \( \Sigma \) \( \text{wff} \) in \( \text{CST}_U^F \), we include \( F(t) \). We establish that \( \text{CST}_U^F \) is a conservative extension of \( \text{CST}_U \).

**Lemma 20 (Soundness)**

There is a translation \( * \) from the syntax of \( \text{CST}_U^F \) to that of \( \text{CST}_U \) such that

1. If \( \Gamma \vdash_{\text{CST}_U^F} \Theta \) then \( \Gamma^* \vdash_{\text{CST}_U} \Theta^* \)
2. If \( c \vdash_{\text{CST}_U} \Theta \) then, where \( \Theta \) does not contain \( R, \Theta^* = \Theta \)
3. If \( \psi \) is \( \Sigma \) in the extended theory, then its translation is \( \Sigma \) in the original.

where \( \Gamma^*, \Theta^* \) are the translated context and judgement.

PROOF. In our raw syntax, we employ De Morgan’s algorithm to push all the negations through to atomic cases. Suppose that \( F \not\equiv [x : I, y : \Theta | \psi] \). Initially, we define \( * \) on the rules of formation to remove just one instance of \( F \). In the atomic terms and their negations, we replace \( F \) as follows.

\[
\alpha[F(t)/y]^* = \exists u : I \cdot \exists v : O \cdot \psi[u, v] \land (t = u) \land \alpha[v/y] \tag{1}
\]

\[
(-\alpha[F(t)/y])^* = \exists u : I \cdot \exists v : O \cdot \psi[u, v] \land (t = u) \land -\alpha[v/y] \tag{2}
\]

The propositional connectives \( \land \) and \( \lor \) are translated compositionally. For the quantifiers, we remove an occurrence of \( F \) in the types via:

\[
(\exists x : T[F(t)/y] \cdot \phi)^* = \exists u : I \cdot \exists v : O \cdot \psi[u, v] \land (u = t) \land \exists x : T[v/y] \cdot \phi \tag{3}
\]

\[
(\forall x : T[F(t)/y] \cdot \phi)^* = \exists u : I \cdot \exists v : O \cdot \psi[u, v] \land (u = t) \land \forall x : T[v/y] \cdot \phi. \tag{4}
\]

An easy induction establishes

\[
\forall x : I \cdot \phi[F[x]]^* \leftrightarrow \exists v : O \cdot \psi[x, v] \land \phi[v] \tag{\heartsuit}
\]

We interpret the other judgements as follows.

\[
c \vdash \Theta[F(t)/x]
\]

translated to the two Judgments

\[
c \vdash t : I \text{ and } c, x : O \vdash \Theta[x]
\]

Part 2 is routine to check. For part 1, for the removal of our single occurrence of \( F \), we prove: if \( \Gamma \vdash_{\text{CST}_U} \Theta \), then \( \Gamma^* \vdash_{\text{CST}_U} \Theta^* \). We illustrate with some exemplary cases. For simplicity, the only occurrence of \( F \) is the one indicated. First, consider the rule.

\[
\frac{a : A \quad b[F(t)/y] : \text{Set}(A)}{a \otimes (b[F(t)/y]) : \text{Set}(A)}
\]
The premise unpacks under the translation to:

\[ a : A \quad t : I \quad y : O \vdash b[y] : Set(A) \]

This yields the translation of the conclusion, i.e.

\[ t : I \quad y : O \vdash a \oplus b[y] : Set(A) \]

This is clearly derivable. The rest of the type inference rules of CST\(_U\) are equally easy to verify. For the logical ones, the quantifier rules are the only non-obvious ones. We illustrate with the following case.

\[
\begin{align*}
\forall x : T \cdot \phi & \quad s[F(t)/y] : T \\
\phi[s[F(t)/x]] & 
\end{align*}
\]

The premises translate to:

\[
\forall x : T \cdot \phi \quad t : I \quad y : O \vdash s[y] : T
\]

By \(\clubsuit\) we must show that

\[
\exists v : O \cdot \psi[t, v] \land \phi[s[v]]
\]

By the assumption of functionality, \(\exists v : O \cdot \psi[t, v]\). Hence, by the translated premises of the rule, \(s[v] : T\). By the quantifier elimination rule we are done. This leaves us to check the new rules. \(F_1\) is immediate. For \(F_2\), we must establish

\[
\forall x : I \cdot \phi[x, F(x)]
\]

Using \(\clubsuit\) and functionality, the result is immediate.

\[
(\forall x : I \cdot \phi[x, F(x)])^* = \forall x : I \cdot \exists u : I \cdot \exists v : O \cdot \psi[u, v] \land u = x \land \psi[u, v]
\]

This provides the soundness of the translation for the removal of one occurrence of \(F\). By iterating this procedure, we eventually remove them all. We then only need to observe that, where there is the only occurrence to remove, the proof of \(\Gamma^* \vdash_{CST_U^*} \Theta^*\), can be carried out in \(CST_U\).

**Theorem 21** (Conservative extension for functions)

Suppose that \(\Gamma, \Theta\) are expressed in the BNF syntax of CST\(_U\). Then

\[
\Gamma \vdash_{CST_U} \Theta \text{ implies } \Gamma \vdash_{CST_U} \Theta
\]

### 4 Normal forms

One of the standard results in elementary recursion theory is the normal form result for computable functions. It guarantees that every partial computable function can be obtained
from a primitive recursive one by one application of minimization. In a similar vein, the
reflection principle for admissible sets [3] states that every $\Sigma$ wff is equivalent to one that has
only one existential quantification over all sets. In this section, we introduce and establish
a parallel result for our theories. It guarantees that every $\Sigma$-specification can be expressed
as an existentially quantified $\Delta_0$ one. We begin with the standard notion.

**Definition 22**
The class of $\Sigma_1$ wff is the smallest class containing the $\Delta_0$ wff and such that:
if $\phi$ is $\Delta_0$ then $\exists x : T \cdot \phi$ is $\Sigma_1$.

For the proof we shall employ the following trick, adapted from [3].

**Definition 23**
Let $\phi$ be any $\Sigma$ wff. Suppose that $w$ is not free in $\phi$ and that $T$ is any term that contains
no type variables that are bound in $\phi$. We then define

$$\phi^{(w, T)}$$

by replacing every occurrence of $\exists x : T \cdot \psi$ in $\phi$ by $\exists x \in w \cdot \psi$.

**Lemma 24**
Let $\phi$ be any $\Sigma$ wff and $T$ any term and $c$ any declaration context such that $c \vdash \phi$ prop
and $c \vdash T$ type. Then:

1. $c \vdash \forall x : \text{Set}(T) \cdot \forall y : \text{Set}(T) \cdot x \subseteq y \rightarrow (\phi^{(x, T)} \rightarrow \phi^{(y, T)})$
2. $c \vdash \forall x : \text{Set}(T) \cdot \phi^{(x, T)} \rightarrow \phi$

**Proof.** By induction on the structure of $\Sigma$ wff, we illustrate the both parts with the
interesting case, i.e. where $\phi$ has the form $\exists z : T \cdot \psi$. Consider 1. By definition,
$(\exists z : T \cdot \psi)^{(x, T)} = \exists z \in x \cdot \psi^{(x, T)}$. By induction, $\exists z \in x \cdot \psi^{(y, T)}$. By the assumption $x \subseteq y$, we
have: $\exists z \in y \cdot \psi^{(y, T)}$. By definition, $(\exists z : T \cdot \psi)^{(x, T)}$. For 2, assume that $x : \text{Set}(T)$. We have to
show $(\exists z : T \cdot \psi)^{(x, T)} \implies \exists z : T \cdot \psi$. Assume the former. By definition, $\exists z \in x \cdot \psi^{(x, T)}$. By
induction, $\exists z \in x \cdot \psi$. Assume $z \in x$ and $\psi$. Since $x \in \text{Set}(T)$, $z : T$, $\exists z : T \cdot \psi$, as required. □

With this property established can now prove the following.

**Proposition 25**
Let $\phi$ be any $\Sigma$ wff and $T$ any term and $c$ any declaration context such that $c \vdash \phi$ prop
and $c \vdash T$ type. Then

$$c \vdash \phi \iff \exists x : \text{Set}(T) \cdot \phi^{(x, T)}$$

where $x \notin FV(\phi)$.

**Proof.** Assume $\exists x : \text{Set}(T) \cdot \phi^{(x, T)}$. By the second part of the previous lemma, $\phi$ follows.
Conversely, we employ induction on the structure of $\Sigma$ wff. We illustrate with the most
important cases. Suppose that $\phi$ has the form $\delta \land \eta$. By induction,

$$\exists x : \text{Set}(T) \cdot \delta^{(x, T)} \quad \exists y : \text{Set}(T) \cdot \eta^{(y, T)}$$
Let $w$ be the union of the guaranteed sets $x, y$. Then, by the first part of the previous lemma, the proof is complete. Now, consider the case where $\phi$ has the form $\forall x \in b \cdot \eta$. Assume that $\forall x \in b \cdot \eta$ and that $x \in b$. Hence, $\eta$ follows. By induction, $\exists y : \text{Set}(T) \cdot \eta^{(v, T)}$. Hence, $\forall x \in b \cdot \exists y : \text{Set}(T) \cdot \eta^{(v, T)}$. Since $c \vdash \phi \text{ prop}$, we have $c \vdash b : \text{Set}(B)$ for some $B$ such that $c \vdash B \text{ type}$. Thus, we may apply collection. So, we have: $\exists z' : \text{Set}(B) \cdot \forall x \in b \cdot \exists y \in z' \cdot \eta^{(v, T)}$. Let $a = \bigcup z'$. By the previous lemma (part 1), $\forall x \in b \cdot \eta^{(a, T)}$. Hence, $\exists w : \text{Set}(T) \cdot \forall x \in b \cdot \eta^{(w, T)}$. Exactly the same strategy works for the numerical quantifiers. Finally, suppose that $\phi$ has the form $\exists x : T \cdot \eta$. Assume $x : T$ and $\eta$. By induction, $\exists y : \text{Set}(T) \cdot \eta^{(v, T)}$. Assume $\eta^{(v, T)}$. Let $w = x \otimes v$. Then $w : \text{Set}(T)$ and $x \in w$ and by the previous lemma (part 1), $\exists x \in w \cdot \eta^{(w, T)}$, as required.

The following simple example illustrate the equivalence:

**Example 26**

$$
\exists x : N \cdot \exists y : \text{Set}(N) \cdot x \in y \leftrightarrow \\
\exists u : \text{Set}(N) \cdot (\exists x : N \cdot \exists y : \text{Set}(N) \cdot x \in y)^{(u, N)} \leftrightarrow \\
\exists u : \text{Set}(N) \cdot \exists x \in u \cdot \exists y : \text{Set}(N) \cdot x \in y
$$

This brings us to the main result.

**Theorem 27**

($\Sigma$ reflection). Let $\phi$ be any $\Sigma$ wff such that $c \vdash \phi$. Then, there is a $\Delta_0$ wff $\psi$ and a term $T$ such that:

$$
c \vdash \phi \leftrightarrow \exists x : T \cdot \psi
$$

where $x$ is not a free variable of $\phi$.

**Proof.** We replace each of the existential type quantifiers in $\phi$ in the following way. Consider any sub-term of the form $\exists x : A \cdot \eta$. The idea is to use the previous result to move the quantifier to the head of the wff. There are two cases. Suppose that $A$ has no variables that are bound in any of its super-terms. Then we replace it as in the definition and obtain $\exists x : \text{Set}(A) \cdot \phi^{(x, A)}$. Now assume that $\phi$ has a sub-term of the form $\beta = \exists z : B \cdot \delta$, where $\delta$ has $\exists x : A[z] \cdot \eta$ as a sub-term. Then, we apply the last proposition to $\beta$ to obtain

$$
\beta \leftrightarrow \exists x : B \cdot \exists y : \text{Set}(A[x]) \cdot \delta^{(y, A[x])}
$$

This yields $\beta \leftrightarrow \exists x : \Sigma x : B \cdot \text{Set}(A[x]) \cdot g^{(x_2, A[x])}$. We have thus moved the bound variables in the type to the head of the term. We repeat this for the whole of $\delta$ so that eventually $\delta$ will contain no occurrences of $\exists x : A \cdot \eta$. We may then begin to process $\exists z : \Sigma x : B \cdot \text{Set}(A[x]) \cdot \delta^{(x_2, A)}$ and move it up to the next block. Eventually, all existential quantifiers will be moved to the head of the term, being replaced by set quantifiers. We then obtain types $T, S$ such that:

$$
c \vdash \phi \leftrightarrow \exists x : T \cdot \phi^{(t[x], S[x])}
$$

where $\phi^{(t[x], S[x])} \in \Delta_0$. 

\[\square\]
EXAMPLE 28

\[ \exists z : U \cdot \exists x : Set(z) \cdot x = x \]

Applying the algorithm we obtain the following equivalencies.

\[ \exists z : U \cdot \exists y : Set(Set(B)) \cdot (\exists x : Set(z) \cdot x = x)^{y, Set(z)} \]
\[ \exists z : U \cdot \exists y : Set(Set(B)) \cdot \exists x \in y \cdot x = x \]
\[ \exists w : \Sigma z : U \cdot Set(Set(B)) \cdot \exists x \in w_2 \cdot x = x. \]

COROLLARY 29

Every schema specification \( R \triangleq [x : T \mid \phi] \) has a normal form \( S \triangleq [x : T \mid \psi] \) where \( \psi \) is \( \Sigma_1 \) and \( x : T \vdash \phi \leftrightarrow \psi. \)

5 Specifications as data

An acid test for any theory of specification must be an account of higher-order specifications, i.e. the specification of relations and functions that treat specifications themselves as data items. To this end, we enrich the theory: let CST\(_{US}\) be the theory obtained from CST\(_U\) by enriching the syntax and rules in the following way.

To the BNF syntax we add two new term structures and a new atomic wff

\[ t ::= \ldots [x : t] \phi \mid S(t) \]
\[ \phi ::= \ldots t(t) \]

i.e. we add schemata and their types to the syntax of terms and a notion of (schema) application as a new atomic wff. In \( \Delta_0 \) wff, only schema of the form \( [x : t \mid \delta] \), i.e. where \( \delta \) itself is \( \Delta_0 \), may occur in atomic wff and their negations. In \( \Sigma \) ones, only schema of the form \( [x : t \mid \sigma] \), where \( \sigma \) is \( \Sigma \), may occur in atomic wff but not in their negations.

With these preliminaries in place, we can now state the schema rules.

\[
\begin{align*}
\text{Sc}_0 & \quad T \text{ type} \\
S(T) & \text{ type} \\
\text{Sc}_1 & \quad x : T \vdash \phi \text{ prop} \\
& \quad [x : T[\phi] : S(T)] \\
\text{Sc}_2 & \quad s : S(T) \quad t : T \\
& \quad s(t) \text{ prop} \\
\text{Sc}_3 & \quad x : T \vdash \phi \text{ prop} \\
& \quad t : T \\
& \quad [x : T[\phi](t) \leftrightarrow \phi[t/x]]
\end{align*}
\]

where in \( \text{Sc}_1 \) and \( \text{Sc}_3 \), \( \phi \) must be \( \Sigma \). \( \text{Sc}_0 \) is the formation rule, \( \text{Sc}_1 \) informs us that a schema whose declaration is of type \( T \) is of type \( S(T) \). \( \text{Sc}_2 \) enables the use of schemata as predicates and \( \text{Sc}_3 \) tells us what they mean. Observe that we have not adopted an axiom of extensionality, i.e. schemata are non-extensional predicates.

All the main results continue to hold including the conservative extension results and the normal form theorem. The logic and the type system still cohere, and there are the obvious extensions to the type-checking theorem.
**Proposition 30 (Type-checking)**

1. $c \vdash s(t) \text{prop} \iff c \vdash t : T$ and $c \vdash s : S(T)$, for some $T$
2. $c \vdash [x : T \mid \phi] : S(T)$ iff $c \vdash T \text{type}$ and $c, x : T \vdash \phi \text{prop}$
3. $c \vdash S(T) \text{type}$ iff $c \vdash T \text{type}$

For our first example to illustrate the use of schemata as data objects, we specify relational composition. Here, schemata represent relations and composition for them is defined in the standard way.

**Example 31 (Relation composition)**

\[
\begin{align*}
\text{Relcomp} \\
{f^?} : S(X \otimes Y), {g^?} : S(Y \otimes Z), {h!} : S(X \otimes Z) \\
\hline
h = [(x, z) : X \otimes Z \mid \exists y : Y \cdot f(x, y) \land g(y, z)]
\end{align*}
\]

The next two provide a perspective on the schema calculus [32]. The first provide the definition of schema conjunction and the second a definition of schema quantification.

**Example 32 (Schema conjunction)**

\[
\begin{align*}
\text{And} \\
{f^?} : S(X \otimes Y), {g^?} : S(Y \otimes Z), {h!} : S(X \otimes Y \otimes Z) \\
\hline
h = [x : X, y : Y, z : Z \mid f(x, y) \land g(y, z)]
\end{align*}
\]

**Example 33 (Hiding)**

\[
\begin{align*}
\text{Exist} \\
{f^?} : S(X \otimes Y \otimes Z), {g!} : S(X \otimes Y) \\
\hline
g = [x : X \otimes Z \mid \exists y : Y \cdot f(x_1, y, x_2)]
\end{align*}
\]
Clearly there are many more simple and complicated examples. But it seems clear that this offers a very expressive and elegant language for the specification of operations that operate over the specifications themselves. However, have we not slipped back into something like HOL? No, for the following reason.

6 Theories of data

We show that our theories have an interpretation in PA. This will provide a consistency proof for them (important for the latter theory) chart their proof theoretic strengths and establish them as theories of data. We shall employ a fair number of standard results about PA with, in most cases, only a reference to where the proofs can be found.

In what follows, the theory PRA (primitive recursive arithmetic) will denote the theory where induction is restricted to \( \Sigma \) propositions [28, 31]. All the constructions we shall employ in interpreting the term language of our theories can be carried out in PRA. We shall actually interpret the whole BNF syntax of CST and interpret type membership as a wff of PRA/PA.

We may conservatively add classes \( \{ x \cdot \phi \} \) to PRA/PA in the standard way. We shall assume some standard primitive recursive bijective coding of the expressions of PA/PRA, including such classes. Indeed, in what follows \( [e] \) will denote the code of the expression \( e \). We may also assume it is bijective and often identify expressions with their codes.

We represent pairs in the standard way as follows.

\[
(x, y) \triangleq \frac{(x + y)^2 + 3x + y}{2}
\]

Pairing is a primitive recursive function. Moreover, it is a matter of arithmetic that this is injective and surjective and that there are primitive recursive functions \( \pi_1, \pi_2 \) that satisfy the standard axioms. We define:

\[
A \otimes B \triangleq \{ (x, y) \cdot x \in A \land y \in B \}
\]

For the representation of finite sets, we require the following result from [15].

- For each \( x, y \), there are unique \( u \leq y, v \leq 1, w \leq 2^x \) such that \( y = u \times 2^{x+1} + v \times 2^x + w \). Then, define

\[
x \in y \triangleq (\text{Bit}(x, y) = 1)
\]

where \( \text{Bit}(x, y) \) is the unique \( v \leq 1 \) such that

\[
\exists u \leq y \cdot \exists w < 2^x \cdot y = u \times 2^{x+1} + v \times 2^x + w
\]

The proof of the following may also be found in [15].

- In PRA, we have:

\[
\forall x \cdot \forall y \cdot \exists z \cdot \forall u \cdot u \in z \leftrightarrow u = x \lor u \in y
\]
This provides a representation of union and hence insertion. Next, we represent the Set constructor. Here, we appeal to the representation of recursive predicates in PA/PRA. First, we add to the language of PA, a unary numerical predicate symbol $X$. The following is a standard result from formal number theory.

For each proposition $\phi$, there is a definable $\theta$ such that

1. $\forall x \cdot \phi[\theta, x] \rightarrow \theta[x]$
2. $[\forall x \cdot \phi[\psi, x] \rightarrow \psi[x]] \rightarrow \forall x \cdot \theta[x] \rightarrow \psi[x]$.

where $\phi[\theta]$ is to be interpreted as $\phi$ with every occurrence of any proposition of the form $\theta[u]$ substituted for $X(u)$. In PRA, $\psi$ must be $\Sigma$.

To construct a representation of the Set constructor, we first apply this result to the proposition $\phi[x] = x = 0 \lor \exists u : A \cdot \exists v : X \cdot x = u \otimes v$. Given the $\theta$ for this $\phi$, we represent the set constructor in the model as: $\text{Set}[A] = \{x \cdot \theta[x]\}$.

The following is also a consequence of the theorem on recursive predicates but a proof almost from scratch can be found in [31].

- (PRA) Let $n > 0$ be given. There is a $\Sigma$-definable relation $\text{Sat}_n(x, x_1, \ldots, x_n)$ such that for each $\Sigma$ wff $\phi$ of PA with exactly the free variables $x_1, \ldots, x_n$ ($n > 0$),

$$\text{Sat}_n([\phi], x_1, \ldots, x_n) \leftrightarrow \phi[x_1, \ldots, x_n]$$

We then define membership in $\Sigma$ classes as follows.

$$x : y \triangleq \text{Sat}_{n+1}([\phi], x, x_1, \ldots, x_n)$$

where $y = [\{x \cdot \phi[x, x_1, \ldots, x_n]\}]$.

We can now proceed to represent schemata and the universe. For schemata, we represent $[x : T \mid \phi]$ as the code of the $\Sigma$ class $\{x \cdot x : T \land \phi\}$ and

$$S(T) = \{n \cdot n \text{ is the code of a } \Sigma \text{ class of the form } \{x \cdot x : T \land \phi\}\}$$

We now interpret all the types of CST$_{US}$ via the codes of their PA class representations, and in particular, $U$ is interpreted as the code of the $\Delta_0$ class that consists of the codes of $\Delta_0$ classes of PA.

**Theorem 34**

CST$_{US}$ is a conservative extension of PA and PRCST is a conservative extension of PRA. Moreover, the $\Sigma$ wff of CST$_{US}$ are interpreted as $\Sigma$ wff of PA.

## 7 Schema definitions

The view we have taken is that $\Sigma$-specifications express the computable relationship that must hold between the inputs and output of an operation. But some constraints that we might wish to apply to specifications are clearly not $\Sigma$. For example, we might wish to insist that
a relation be total or functional. However, such non-computable constraints cannot be, in any syntactic sense, part of any program that meets the specification. Rather, they are meta-assertions that describe the mathematical properties of any such program. Moreover, such non-computable properties of programs can be expressed in the language of our theories, and so, in principle, we need no further notation. Nevertheless, it might be thought convenient to extend our schema notation to facilitate the expression of such constraints, i.e. allow

\[ R \triangleq [x : T | \phi] \]

where \( \phi \) may not be \( \Sigma \). To distinguish matters, we shall call these *schema definitions* rather than schema specifications. Note that, we are not changing the theory; the resulting system is a conservative extension. Most significantly, schema types still only contain \( \Sigma \) schema. Indeed, within the theory, schema definitions have no type.

To illustrate their possible use we provide a schema definition of the actual notion of function and a definition of Cauchy sequences.

**Example 35 (Functions)**

\[
\begin{array}{c}
Fun \\
\hline
f : S(X \otimes Y) \\
\forall x : X \quad \exists ! y : Y \cdot f(x, y) \\
\end{array}
\]

**Example 36 (Cauchy sequences)**

\[
\begin{array}{c}
Real \\
\hline
f : S(N \otimes Q) \\
\exists z : N \cdot \forall m, n \geq z \cdot \forall u, v : Q \cdot (f(m, u) \wedge f(n, v)) \rightarrow |u - v| \leq 1/z \\
\end{array}
\]

where \( Q \) is the type \( N \otimes N \).

---

9. This makes it different to approaches that seek to restrict the whole language of the theory, e.g. geometric logic [37].
With schemata as data items, these express non-computable properties that they may possess. Following on from this, the following is a definition of addition for these sequences.

**Example 37 (Real addition)**

\[ f : S(N, Q), g : S(N, Q), h : S(N, Q) \]

\[
\begin{align*}
    h &= [(x, y) : N \otimes Q \mid \exists z_1 : Q \cdot \exists z_2 : Q \cdot f(2x, z_1) \wedge g(2x, z_2) \wedge y = z_1 + z_2] \\
    &\quad \land \text{Real}(f) \wedge \text{Real}(g) \wedge \text{Real}(h)
\end{align*}
\]

Such schema definitions may be used to aid the mathematical activity associated with specification. For example, the last operation succinctly demands that the underlying computable operation preserves the reals. Indeed, such definitions might be employed in the actual specification process to guide the design of specifications. However, ultimately they must fall by the wayside. In the formal paradigm for program development, programs are obtained from specifications via refinement, in the sense of [29]: an operation \( S \) is a refinement of another \( R \) if the domain of \( S \) is at least as big as that of \( R \) and on the domain of \( R \) \( S \) behaves identically. The point of adding such non-computable constraints is to inform the specification designer about the intended properties of the resulting system. While these properties may constrain the system in non-computational ways, somewhere on the way to program development, they must be refined away. For example, although real addition is not \( \Sigma \), it is possible to refine it to a \( \Sigma \)-specification—simply drop the Real constraints in the predicate.

Of course, not all schema definitions may be refined to \( \Sigma \) ones. Although it could be more precisely coded, it should be clear that, being total, if the following could be refined to a \( \Sigma \) definition, we would be able to effectively solve the halting problem.

**Example 38 (K)**

\[
K
\]

\[
\begin{align*}
    i? : N \\
    o! : N
\end{align*}
\]

\[
\begin{align*}
    T_i(i) \downarrow \land o = 0 \\
    \lor \\
    \neg (T_i(i) \downarrow) \land o = 1
\end{align*}
\]

where \( T_i(i) \downarrow \) says the Turing machine with code \( i \) halts on input \( i \).
While schema definitions are a mathematical luxury rather than a necessity, they do clarify, and make more convenient, the use and the role of non-computable properties in specification. Finally, be aware that we are discussing the non-computational properties of our notion of schema; we are not claiming anything about operations as data items [5]. Indeed, a topic worthy of exploration would be to enrich the theory with a new type of operations; but that is a topic for another occasion.

References


Received 2 August 2005