
The Foundations of Specification

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

Abstract

We develop and explore a *Core Specification Theory (CST)* as a basis for the meta-mathematical investigation of specification and specification languages.

Keywords: Scheme, conservative extensions, specification.

1 Specification theories

Specification languages such as **Z**, **VDM**, **B**, **PVS** [32, 14, 22]¹, [1, 8, 26, 27]² are based upon the Predicate Calculus (**PC**) in the sense that the language in which the specifications are expressed is some version or dialect of **PC**. However, they differ from the simple *single sorted* versions of **PC** in that they distinguish between different *types* of data, where the *types* of these theories are usually presented in an inductive fashion: there are basic types (e.g. *numbers*, *characters*) together with a battery of type constructors such as *products*, *sets* and *recursive types*. Furthermore, presentations of specification languages [21, 13, 16, 18, 10, 11, 23, 24, 31, 5, 20, 4, 6, 9, 19, 26] sometimes include groups of axioms for the various types and their associated relation and function symbols. Typically these stipulate the membership conditions for the type, the criteria for two elements of the type to be equal and lay out the properties of any special relations and functions for the type. As such these axiomatic systems constitute *theories* of the underlying concepts of the language that we shall refer to as *Specification Theories*.

However, it is not an easy task to provide such axiomatizations for existing languages [5, 16, 10, 11]. This is largely due to the way in which they have evolved: their form has been largely determined only by practical considerations. In practice, standard logical systems such as **HOL** or **ZF** have been lifted wholesale and employed as the host for a massive infrastructure of *syntactic sugar* dictated by practical needs. Unfortunately, this *sugaring* has taken over and obscured the logical core. Subsequently, we have languages for which it is virtually impossible to provide mathematically tractable and usable axiomatizations. In particular, the axiom systems are often very large: a stab is made at providing axioms for every construct that might prove useful. Even so, the systems are often incomplete in the sense that some of the constructs are not axiomatized. Indeed, some of the more rigorous attempts at developing axiomatizations for existing languages (e.g. [10, 11]) have been forced to part company with the target language for technical and conceptual reasons. More generally, many problems with current languages often only come to the fore when axiomatizations are attempted.

¹Strictly speaking, PVS is a whole system rather than just a language.

²We do not include constructive theories such as Nuprl, Constructive Type Theory and the Calculus of Constructions, in our sweep of theories. There are several reasons for this. First, the style of specification is based upon extracting programs from constructive proofs; essentially from proofs of $\forall\exists$ statements. This is very different to the approach taken in the classical setting and so would require a much longer paper to treat them. Moreover, more often than not, these theories are more rigorously presented. So some of the criticism aimed at the languages considered here, does not apply.

Indeed, current specification languages are rarely precisely and completely formulated as axiomatic theories and so are inadequate for metamathematical purposes; it is very hard to treat the theory as an object of study in its own right. As a result, our mathematical grasp of specification languages and specification in them, is quite meagre.

This is the first of a series of papers which seek to address this foundational gap. Our objective in this first paper is to formulate and study a core *Specification Theory* and use it to explore the specification process. This theory is a sub-theory of the implicit theories of all the major specification languages; it is buried inside them, even if it is not evidently so. Nevertheless, it is expressive enough to illustrate the different styles of specification employed by these languages, and to explore the logical foundations of the actual process of specification.

2 A core specification theory (CST)

We present a core specification theory (CST) which is a fragment of most, if not all, the major specification languages. We shall do so in several stages. Initially, we present the language, and compared with actual specification languages, it is very small. We then develop the logic of the system: a version of a typed predicate logic. Finally, we present the rules and axioms for the various types.

2.1 The syntax of CST

The language has three syntactic categories: *wff*, *types* and *terms*. We deal first with the syntax for the types since these drive the form of the language. The atomic type terms consist of type variables and the type constant N , the *natural number* type. There are two type constructors that permit the formation of *sets* (Set) and *Cartesian products* (\otimes). More formally, the syntax of type terms is given as follows.

$$T ::= X \mid N \mid T \otimes T \mid Set(T) .$$

These types are taken as basic in both Z and VDM. Generally, we shall employ upper case Roman letters A, B, C, D, \dots for type terms with U, V, W, X, Y, Z reserved for type variables.

With these go the following individual term constants and function symbols. Apart from the individual variables we admit, for the natural numbers, the constant zero (0) and the numerical successor function ($^+$) and for Cartesian products, we include the pairing function ($()$) and the selection functions (π_i). Finally, sets are supported by a constant for the empty set (\emptyset) and a binary *insertion* function (\otimes) for adding an element to a set. This leads to the following syntax for terms.

$$\begin{aligned} t ::= & x \mid 0 \mid t^+ \\ & (t, t) \mid \pi_1(t) \mid \pi_2(t) \\ & \emptyset \mid t \otimes t \end{aligned}$$

where generally we employ lower case Roman letters a, b, c, \dots for individual terms with u, v, w, x, y, z reserved for term variables. The basic operators of the theory are *polymorphic*. In particular, the set insertion function and the pairing operation operate globally over all types.

Finally, we introduce the well-formed formula (wff). We employ lower case Greek letters for these. The atomic wff include absurdity (Ω); equality, set membership (\in) and the ordering relation on the natural numbers ($<$). General wff are generated from these by the propositional connectives

and the quantifiers.

$$\begin{aligned}
\phi &::= \Omega \mid t = t \mid t \in t \mid t < t \\
&\neg\phi \mid \phi \wedge \phi \mid \phi \vee \phi \mid \phi \rightarrow \phi \\
&\forall x < t \cdot \phi \mid \exists x < t \cdot \phi \\
&\forall x \in t \cdot \phi \mid \exists x \in t \cdot \phi \\
&\forall x : T \cdot \phi \mid \exists x : T \cdot \phi \\
&\forall X \cdot \phi \mid \exists X \cdot \phi .
\end{aligned}$$

Apart from the numerical and set quantifiers, we have quantification with respect to a given type and quantification over types. The last four are the main logical quantifiers of the theory and will be governed by standard introduction and elimination rules. One might think that the bounded quantifiers should be defined in terms of the others but, for theoretical and practical purposes, it is convenient to take them all as primitive. However, their properties will be stated in terms of the main quantifiers.

Where e is a term or wff, we shall write $FV(e)$ for the collection of free individual variables of e and $FTV(e)$ for the free type variables. For the purposes of substitution, we shall write $e[x_1, \dots, x_n]$ to mark free individual variables. This notation is not to be taken to imply that all of the variables x_1, \dots, x_n occur free in e nor that they exhaust all the free variables of e . We shall write $e[t_1, \dots, t_n/x_1, \dots, x_n]$ for the meta-operation of substituting the terms t_i for the variables x_i . Similarly, we shall write $e[X_1, \dots, X_n]$ to mark type variables and $e[T_1, \dots, T_n/X_1, \dots, X_n]$ for type substitution. Finally, note that individual terms do not contain type variables and type terms do not contain individual variables.

We next present a few preliminary definitions. Propositional equivalence (\leftrightarrow) is defined in terms of implication in the standard way. We define *type membership* and some other useful forms of quantification, as follows:

$$\begin{aligned}
t : T &\triangleq \exists x : T \cdot t = x \\
\exists! x : T \cdot \phi[x] &\triangleq \exists x : T \cdot \phi[x] \wedge \forall y : T \cdot \phi[y] \rightarrow x = y \\
\exists^{\leq 1} x : T \cdot \phi[x] &\triangleq \forall x : T \cdot \forall y : T \cdot \phi[x] \wedge \phi[y] \rightarrow x = y .
\end{aligned}$$

This completes all the syntactic preliminaries.

2.2 The logic

The logic is presented in a sequent-style natural deduction format. The rules are given relative to a *context* Γ which is a (possibly empty) finite set of wff. Sequents thus take the form:

$$\Gamma \vdash_{\text{CST}} \phi$$

which is to be understood as asserting that in the theory **CST**, ϕ follows from Γ . We shall usually drop the subscript. Furthermore, we shall only include the contexts of a rule where they are modified in passing from the premisses to the conclusion. We shall also write rules with no premisses in the standard way.

There are two *structural* rules: an *assumption* axiom and a *weakening* rule.

$$\mathbf{Ax} \quad \phi \vdash \phi \qquad \mathbf{W} \quad \frac{\Gamma \vdash \psi}{\Gamma, \phi \vdash \psi}$$

There are the two standard equality axioms — adapted to a typed setting.

$$\begin{aligned} \mathbf{E}_1 \quad & \forall x : X \cdot x = x \\ \mathbf{E}_2 \quad & \forall x : X \cdot \forall y : X \cdot x = y \rightarrow (\phi[x] \rightarrow \phi[y]) \end{aligned}$$

The *logical* rules are the normal classical ones. We begin with the propositional connectives.

$$\begin{array}{lll} \mathbf{L}_1 \quad \frac{\neg\phi \quad \phi}{\Omega} & \mathbf{L}_2 \quad \frac{\Omega}{\phi} & \\ \mathbf{L}_3 \quad \frac{\Gamma, \phi \vdash \Omega}{\Gamma \vdash \neg\phi} & \mathbf{L}_4 \quad \frac{\Gamma, \neg\phi \vdash \Omega}{\Gamma \vdash \phi} & \\ \mathbf{L}_5 \quad \frac{\phi \quad \psi}{\phi \wedge \psi} & \mathbf{L}_6 \quad \frac{\phi \wedge \psi}{\phi} & \mathbf{L}_7 \quad \frac{\phi \wedge \psi}{\psi} \\ \mathbf{L}_8 \quad \frac{\phi}{\phi \vee \psi} & \mathbf{L}_9 \quad \frac{\psi}{\phi \vee \psi} & \mathbf{L}_{10} \quad \frac{\Gamma \vdash \phi \vee \psi \quad \Gamma, \phi \vdash \eta \quad \Gamma, \psi \vdash \eta}{\Gamma \vdash \eta} \\ \mathbf{L}_{11} \quad \frac{\Gamma, \phi \vdash \psi}{\Gamma \vdash \phi \rightarrow \psi} & \mathbf{L}_{12} \quad \frac{\phi \rightarrow \psi \quad \phi}{\psi} & \end{array}$$

The main quantifier rules are for the typed and type quantifiers. These are standard and subject to the normal side conditions about dependency, e.g. in \mathbf{L}_{16} , x must not be free in any wff in Γ and η .

$$\begin{array}{ll} \mathbf{L}_{13} \quad \frac{\Gamma, x : X \vdash \phi}{\Gamma \vdash \forall x : X \cdot \phi} & \mathbf{L}_{14} \quad \frac{\forall x : X \cdot \phi \quad t : X}{\phi[t/x]} \\ \mathbf{L}_{15} \quad \frac{\phi[t/x] \quad t : X}{\exists x : X \cdot \phi} & \mathbf{L}_{16} \quad \frac{\Gamma \vdash \exists x : X \cdot \phi \quad \Gamma, x : X, \phi \vdash \eta}{\Gamma \vdash \eta} \\ \mathbf{L}_{17} \quad \frac{\phi}{\forall X \cdot \phi} & \mathbf{L}_{18} \quad \frac{\forall X \cdot \phi}{\phi[T/X]} \\ \mathbf{L}_{19} \quad \frac{\phi[T/X]}{\exists X \cdot \phi} & \mathbf{L}_{20} \quad \frac{\Gamma \vdash \exists X \cdot \phi \quad \Gamma, \phi \vdash \eta}{\Gamma \vdash \eta} \end{array}$$

We shall deal with the bounded quantifiers in connection with their types. This completes the basic logic of the theory. We can now deal with the types themselves: for each we shall provide introduction, elimination and special equality rules together with any rules for their special relations and functions.

2.3 *Natural numbers*

The first group of axioms for the numbers are those of *Peano* Arithmetic but with explicit quantifiers to restrict them to numbers. The first four inform us about the successor relation and the fifth is the

standard scheme of induction.

$$\begin{aligned}
\mathbf{N}_1 & 0 : N \\
\mathbf{N}_2 & \forall x : N \cdot x^+ : N \\
\mathbf{N}_3 & \forall x : N \cdot x^+ \neq 0 \\
\mathbf{N}_4 & \forall x : N \cdot \forall y : N \cdot x^+ = y^+ \rightarrow x = y \\
\mathbf{N}_5 & (\phi[0] \wedge \forall x : N \cdot \phi[x] \rightarrow \phi[x^+]) \rightarrow \forall x : N \cdot \phi[x] .
\end{aligned}$$

The next group provide the axioms for the ordering relation. Again, they are the standard axioms adapted to fit the present typed framework.

$$\begin{aligned}
\mathbf{N}_6 & \forall x : N \cdot \neg(x < 0) \\
\mathbf{N}_7 & \forall y : N \cdot \forall x : N \cdot x < y^+ \leftrightarrow (x < y \vee x = y) .
\end{aligned}$$

Finally, we deal with the bounded numerical quantifiers. These are governed by the following axioms.

$$\begin{aligned}
\mathbf{N}_8 & \forall y : N \cdot (\forall x < y \cdot \phi) \leftrightarrow (\forall x : N \cdot x < y \rightarrow \phi) \\
\mathbf{N}_9 & \forall y : N \cdot (\exists x < y \cdot \phi) \leftrightarrow (\exists x : N \cdot x < y \wedge \phi) .
\end{aligned}$$

They insist that, in the context where the bound is a number, they can be unpacked in terms of quantification with respect to the natural number type. This style of axiom, where a construct is only provided a meaning in a given context, will form the basis for our general scheme of relation and function specifications. This completes the numerical axioms and rules.

2.4 Cartesian products

Cartesian products are present in most specification languages and the axioms are the usual ones. The first three are the normal axioms for pairs and selection functions.

$$\begin{aligned}
\mathbf{P}_1 & \forall x : X \cdot \forall y : Y \cdot (x, y) : X \otimes Y \\
\mathbf{P}_2 & \forall z : X \otimes Y \cdot \pi_1(z) : X \\
\mathbf{P}_3 & \forall z : X \otimes Y \cdot \pi_2(z) : Y .
\end{aligned}$$

Finally, the special equality axioms demand that the selection functions behave appropriately on pairs and support surjective pairing.

$$\begin{aligned}
\mathbf{P}_4 & \forall x : X \cdot \forall y : Y \cdot \pi_1(x, y) = x \wedge \pi_2(x, y) = y \\
\mathbf{P}_5 & \forall z : X \otimes Y \cdot z = (\pi_1(z), \pi_2(z)) .
\end{aligned}$$

The construction can be iterated to the product of more than two types via

$$A_1 \otimes (A_2 \otimes \dots \otimes A_{n+1}).$$

In particular, we shall write A^n for $A \otimes A \otimes \dots \otimes A$ i.e. n -copies of A . We shall often write $\pi_i(x)$ as x_i .

2.5 Sets

While it is more central in some than in others, this type constructor is present in some form in most specification languages and certainly in all the major logic-based languages. We present the axioms/rules in several waves.

The first group parallel the Peano axioms. The first pair state the closure conditions for the type: the empty set is a member of every type of sets and the sets of a given type are closed under element insertion. The next two ban the multiplicity of elements in sets and guarantee order independence. The final axiom in the group is the induction principle for sets.

$$\mathbf{S}_1 \quad \emptyset : \text{Set}(X)$$

$$\mathbf{S}_2 \quad \forall x : X \cdot \forall y : \text{Set}(X) \cdot x \otimes y : \text{Set}(X)$$

$$\mathbf{S}_3 \quad \forall x : X \cdot \forall y : \text{Set}(X) \cdot x \otimes (x \otimes y) = x \otimes y$$

$$\mathbf{S}_4 \quad \forall x : X \cdot \forall y : X \cdot \forall z : \text{Set}(X) \cdot x \otimes (y \otimes z) = y \otimes (x \otimes z)$$

$$\mathbf{S}_5 \quad (\phi[\emptyset] \wedge \forall x : X \cdot \forall y : \text{Set}(X) \cdot \phi[y] \rightarrow \phi[x \otimes y]) \rightarrow \forall y : \text{Set}(X) \cdot \phi[y] .$$

The next pair govern the special relation symbol for this type, namely *set membership*. The first insists that the empty set has no elements and the second demands that the insertion function adds a single element to an existing set.

$$\mathbf{S}_6 \quad \forall x : X \cdot x \notin \emptyset$$

$$\mathbf{S}_7 \quad \forall y : \text{Set}(X) \cdot \forall x : X \cdot \forall z : X \cdot z \in x \otimes y \leftrightarrow (z = x \vee z \in y) .$$

Finally, we provide the *set quantifier* axioms. They mirror the numerical ones and insist that where the bound is a set they can be unpacked in terms of the main type quantifier.

$$\mathbf{S}_8 \quad \forall y : \text{Set}(X) \cdot (\forall x \in y \cdot \phi) \leftrightarrow (\forall x : X \cdot x \in y \rightarrow \phi)$$

$$\mathbf{S}_9 \quad \forall y : \text{Set}(X) \cdot (\exists x \in y \cdot \phi) \leftrightarrow (\exists x : X \cdot x \in y \wedge \phi) .$$

This completes the statement of the theory **CST**. It is a very minimal theory of *numbers, sets and products* with very little meat on it. On the other hand, it is a highly expressive theory which supports a large portion of everyday specification.

2.6 First steps

We establish a few preliminary properties of the theory. The first couple present some elementary properties of the numbers.

PROPOSITION 2.1

The following are provable.

1. $\forall y : N \cdot \forall x < y \cdot x : N$
2. $\forall y : N \cdot y = 0 \vee \exists u : N \cdot y = u^+$.

PROOF. The first follows from N_8 and the second by numerical induction with the induction wff

$$\phi[y] = y = 0 \vee \exists u : N \cdot y = u^+ .$$

■

We now do much the same for sets, but here there are a few more obvious things to say.

PROPOSITION 2.2

The following are provable

1. $\forall y : \text{Set}(X) \cdot \forall x \in y \cdot x : X$
2. $\forall y : \text{Set}(X) \cdot y = \emptyset \vee \exists u : X \cdot \exists v : \text{Set}(X) \cdot y = u \otimes v$
3. $\forall y : \text{Set}(X) \cdot \forall x : X \cdot x \otimes y \neq \emptyset$
4. $\forall y : \text{Set}(X) \cdot \forall x \in y \cdot x \otimes y = y$
5. $\forall z : \text{Set}(X) \cdot \forall x \in z \cdot \exists y : \text{Set}(X) \cdot x \notin y \wedge z = x \otimes y$
6. $\forall x \in \text{Set}(X) \cdot (\forall y \in x \cdot y : Y) \rightarrow x : \text{Set}(Y)$.

PROOF. The first follows from S_8 . The rest employ the obvious set inductions. For example, for (2), we use set induction with the induction wff:

$$\phi[y] = y = \emptyset \vee \exists u : X \cdot \exists v : \text{Set}(X) \cdot y = u \otimes v.$$

We can now establish the most important property of the *sets* of the theory namely their *extensional* nature. We first define *Extensional Equality* for sets as follows.

$$x \equiv y \triangleq \forall u \in x \cdot u \in y \wedge \forall v \in y \cdot v \in x.$$

PROPOSITION 2.3 (Extensionality)

$$\forall x : \text{Set}(X) \cdot \forall y : \text{Set}(X) \cdot x \equiv y \rightarrow x = y.$$

PROOF. Let $x : \text{Set}(X)$. We employ induction on y with the induction wff:

$$\phi[y] = x \equiv y \rightarrow x = y.$$

Assume $y = \emptyset$. If $x = \emptyset$ we are finished by E_1 . But if $x \neq \emptyset$ then $x = u \otimes v$ for some u, v which is impossible. This completes the base case. So assume that $y = u \otimes v$. We have to show that

$$x \equiv u \otimes v \rightarrow x = u \otimes v.$$

If $u \in v$ we are finished by induction. If $u \notin v$, since $u \in x$, by the last proposition (part 5), $x = u \otimes v'$ for some set v' , $u \notin v'$. Since $u \otimes v \equiv u \otimes v'$ and $u \notin v$ and $u \notin v'$, it follows that $v \equiv v'$. By induction, $v = v'$. So $u \otimes v = u \otimes v'$. ■

This completes our basic introduction to the theory **CST**. It should be clear that this theory is a sub-theory of both the implicit theories of **VDM** and **Z** — and indeed all the major languages.

2.7 Related work

In developing **CST** we have been influenced by Hereditary Finite Set Theory (**HFST**). In particular, in unpublished work, Hodges has developed an approach to specification based upon **HFST**. But he does not really consider the impact of types; the present theory may be considered as an attempt to explicitly put types into **HFST** — with all that entails (e.g. the need to take products as primitive, etc.). The present work has also been partly inspired by the work on *Bounded Set Theory* [25], although here the inspiration is more marginal. In [28], Struth develops an algebraic approach to a form of finite set theory but without any types. Connections here are less obvious, but the infrastructure developed may prove to be useful.

3 Relation specification

Implicit in all *logical* specification languages is the notion that specifications involve the introduction of new *relation* and *function* symbols. Furthermore, most languages allow a style of specification in which new *polymorphic* or *generic* relations and functions can be introduced. However, different specification languages present specifications in different syntactic forms. In particular, some (e.g. Z) are predominately relational in their style of specification and others (e.g. VDM) are more functionally inclined. We shall consider both styles. Indeed, we shall provide, within the formal framework provided by CST, a uniform logical foundation for both Z and VDM specification styles.

3.1 Schema

Our style of relation specification is based upon the Z *schema* notation: it introduces new *polymorphic* relation symbols into the language via the following specification format. More exactly, relational specifications take the following shape.

DEFINITION 3.1

Let ϕ be any wff and A_1, \dots, A_k any type terms where x_1, \dots, x_k are all the free individual variables of ϕ and where X_1, \dots, X_n ($n, k \geq 0$) include and exhaust all the type variables of ϕ , A_1, \dots, A_k . We may then introduce a new relation symbol into the language via

$$R[X_1, \dots, X_n] \triangleq [x_1 : A_1, \dots, x_k : A_k \mid \phi] \quad (\mathbf{S})$$

where each free individual variable is assigned exactly one type. We shall call these **schema specifications**. The type prefix

$$x_1 : A_1, \dots, x_k : A_k$$

we call the **declaration** of the schema and the wff ϕ its **predicate**.

How are these specifications to be unpacked *logically*? Here we shall be guided by the form of the axioms of our theory CST: **S** is to be understood as the introduction of a new relation symbol R that satisfies the following axiomatic condition.

$$\forall X_1 \dots \forall X_n \cdot \forall x_1 : A_1 \dots \forall x_k : A_k \cdot R[X_1, \dots, X_n](x_1, \dots, x_k) \leftrightarrow \phi. \quad (\mathbf{Rel})$$

This implicitly extends the syntax of wff to include new atomic wff of the form $R[T_1, \dots, T_n](t_1, \dots, t_k)$.

For the rest of this section we shall employ this notion of specification to enrich the theory. Indeed, the development of the theory will furnish us with material to illustrate the whole specification process.

EXAMPLE 3.2

A schema specification of the **Subset** relation is given as follows:

$$\subseteq_X \triangleq [x : \text{Set}(X), y : \text{Set}(X) \mid \forall z \in x \cdot z \in y],$$

which is written in its standard infix notation with the type variable as a subscript. Under the government of Rel, this is interpreted as the introduction of a new relation which satisfies

$$\forall X \cdot \forall x : \text{Set}(X) \cdot \forall y : \text{Set}(X) \cdot x \subseteq_X y \leftrightarrow \forall z \in x \cdot z \in y.$$

The following are instances.

$$\subseteq_N \triangleq [x : \text{Set}(N), y : \text{Set}(N) \mid \forall z \in x \cdot z \in y]$$

$$\subseteq_{N^2} \triangleq [x : \text{Set}(N^2), y : \text{Set}(N^2) \mid \forall z \in x \cdot z \in y].$$

EXAMPLE 3.3

With subset in place we can specify a generic version of **Extensional Equivalence** for sets as follows:

$$\equiv_X \triangleq [x : Set(X), y : Set(X) \mid x \subseteq_X y \wedge y \subseteq_X x].$$

EXAMPLE 3.4

The following provides the specification of the **Pairing** relation on sets:

$$Pair_X \triangleq [x : X, y : X, z : Set(X) \mid \forall u : X \cdot u \in z \leftrightarrow u = x \vee u = y].$$

Of course, just positing a relation is not the end of the story. We might, for instance, wish to show that the relation is not vacuous. In most cases this will be obvious, but we shall often investigate matters more thoroughly.

Following Z [32], we shall also write schema in the more graphic form:

$$\begin{array}{|l} R[X_1, \dots, X_n] \\ \hline x_1 : T_1, \dots, x_k : T_k \\ \hline \phi \end{array}$$

In this presentation we shall often mark conjunctions with a new line. The following examples illustrate this.

EXAMPLE 3.5

The following is a schema specification of simple set theoretic **union**.

$$\begin{array}{|l} Union[X] \\ \hline u : Set(X), v : Set(X), w : Set(X) \\ \hline \forall x : X \cdot x \in w \leftrightarrow x \in u \vee x \in v \end{array}$$

This is clearly non-vacuous since choosing all three sets to be the empty set provides an instance.

EXAMPLE 3.6

The following provides the definition of **generalized union**

$$\begin{array}{|l} Genunion[X] \\ \hline u : Set(Set(X)), v : Set(X) \\ \hline \forall x : X \cdot x \in v \leftrightarrow \exists z \in u \cdot x \in z \end{array}$$

EXAMPLE 3.7

Given the specification of subset, the **Power set** relation may be specified as follows.

$$\begin{array}{l}
 \text{Pow}[X] \\
 \hline
 u : \text{Set}(X), v : \text{Set}(\text{Set}(X)) \\
 \hline
 \forall x : \text{Set}(X) \cdot x \in v \leftrightarrow x \subseteq_X u
 \end{array}$$

Observe that there is an instance: choose both to be the empty set.

We now come to the way of forming sets given by a scheme of *separation*. Notice that this operation is schematic (in the standard sense) with respect to a wff, i.e. we introduce a new relation symbol for each wff.

EXAMPLE 3.8

Let ψ contain at most z free. We then specify

$$\begin{array}{l}
 \text{Sep}_\psi[X] \\
 \hline
 u : \text{Set}(X), v : \text{Set}(X) \\
 \hline
 \forall z : X \cdot z \in v \leftrightarrow z \in u \wedge \psi[z]
 \end{array}$$

EXAMPLE 3.9

The following provide specifications of the **domain** and **range** of set-theoretic relations.

$$\begin{array}{l}
 \text{Dom}[X, Y] \\
 \hline
 u : \text{Set}(X \otimes Y), v : \text{Set}(X) \\
 \hline
 \forall x : X \cdot x \in v \leftrightarrow \exists y : Y \cdot (x, y) \in u
 \end{array}$$

$$\begin{array}{l}
 \text{Ran}[X, Y] \\
 \hline
 u : \text{Set}(X \otimes Y), v : \text{Set}(X) \\
 \hline
 \forall y : Y \cdot y \in v \leftrightarrow \exists x : X \cdot (x, y) \in u
 \end{array}$$

Although there is no type of *maps* in the present theory, we can specify the relation of being a *map* from one type to a second, i.e. a *set theoretic relation* (an element of $\text{Set}(X \otimes Y)$) which is *single-valued*.

EXAMPLE 3.10

$$Map[X, Y] \triangleq [z : Set(X \otimes Y) \mid \forall x \in z \cdot \forall y \in z \cdot x_1 = y_1 \rightarrow x_2 = y_2]$$

The next example is slightly different in that it is an example of how one can build new specifications uniformly from old ones. We shall have more to say about this in a later publication where we introduce our interpretation of the *schema calculus*.

DEFINITION 3.11

Let

$$R[X_1, \dots, X_n] \triangleq [x : I, y : O \mid \psi].$$

Define the **domain** and **range** of R as follows.

$$\begin{aligned} Dom_R[X_1, \dots, X_n] &= [x : I \mid \exists y : O \cdot \psi] \\ Ran_R[X_1, \dots, X_n] &= [y : O \mid \exists x : I \cdot \psi]. \end{aligned}$$

We shall study these and related examples in some detail since they will be employed to illustrate the whole process of specification. Moreover, many of them will provide some of the central infrastructure for the development of the theory as a more realistic specification language.

Operations express relationships between named *inputs* and *outputs*. However, the specification of an *operation* will have the same form as a schema definition but some of the variables will be interpreted as *inputs* and some as *outputs*. However, apart from the binary case of relations, where the convention is that the first argument is the input and the second the output, we have no convention to determine which is which. We shall often use Cartesian product types to reduce matters to the binary case and then employ this convention. However, in practical applications of the theory, this often proves inconvenient and so we need some more general notational devices to distinguish inputs from outputs. We shall adopt several styles of convention. We shall often just stick the inputs on the first line of the declaration and the outputs on the second. More general Z style conventions employ decorations, e.g. mark inputs with ? and outputs with !. Where there is no danger of ambiguity we shall usually drop the decoration in the predicate.

EXAMPLE 3.12

We may introduce **generalized intersection** as follows.

$$\begin{array}{|l} \cap[X] \\ \hline u? : Set(Set(X)), v! : Set(X) \\ \hline \forall x : X \cdot x \in v \leftrightarrow \forall w \in u \cdot x \in w \end{array}$$

This is our *interpretation* of the Z *schema notation*. However, we are not claiming that it is the official one; we are merely borrowing the schema notation as a convenient way of expressing our style of polymorphic definition. Indeed, there are some obvious differences. First, we have explicitly interpreted schema to herald the axiomatic introduction of new relation symbols. This perspective is not explicitly adopted in the accounts of Z. Second, in the more formal accounts of Z given in the various standardization documents [5, 21], and the various logics of Z [31, 10, 11], it appears

that a schema is taken to imply the type of its arguments. In the present setting this amounts to the introduction of a new relation via the following axiomatic stipulation.

$$\forall X_1 \dots \forall X_n \cdot R[X_1, \dots, X_n](x_1, \dots, x_k) \leftrightarrow x_1 : A_1 \wedge \dots \wedge x_k : A_k \wedge \phi.$$

However, within the present theory, this is not a significant difference since the declaration will always form part of the proof context within which reasoning about the specification will be carried out. This perspective is the one adopted in the statement of the theory.

3.2 *Conservative extensions*

Every time we introduce a new relational symbol via S we enrich the language and the theory. Moreover, we do so by introducing a new *axiom* into the theory. However, we require that such relational enrichment does not substantially change the theory. This would be unfortunate since then we would have no guarantee that the properties of the old theory, which we may rely on during specification, are maintained. Indeed, we shall use the word *legitimate* to describe such conservative additions. This will mean different things within different styles of specification.

Suppose that we have extended the language of \mathbf{CST} with a new relation R . Let \mathbf{CST}^R be the theory in which all the rules of \mathbf{CST} are extended to this new language together with the axiom Rel . The following informs us that such additions are conservative.

THEOREM 3.13

Let ψ be any wff of \mathbf{CST} . Then

$$\text{if } \Gamma \vdash_{\mathbf{CST}^R} \psi \text{ then } \Gamma \vdash_{\mathbf{CST}} \psi.$$

The theorem follows as a direct result of the following lemma which shows how to compile any wff of the extended theory to one of the original.

LEMMA 3.14

There is a translation $*$ from the language of \mathbf{CST}^R to the language of \mathbf{CST} such that

1. if $\Gamma \vdash_{\mathbf{CST}^R} \phi$ then $\Gamma^* \vdash_{\mathbf{CST}} \phi^*$
2. if ϕ is a wff of \mathbf{CST} then $\phi^* = \phi$

where Γ^* is the translated context.

PROOF. We shall spell out the details of the translation since it will be employed as a basis for several modifications. The translation is defined by recursion on the extended language. The major impact is obviously on the new relation symbol. We illustrate with the case of one type variable and one individual argument.

$$R[X] \triangleq [x : A \mid \phi].$$

This is transformed as follows.

$$R[T](t)^* = \phi[T/X, t/x].$$

All the other atomic wff translated to themselves

$$\alpha^* = \alpha.$$

The translation passes through the propositional connectives and the quantifiers, e.g.

$$\begin{aligned}(\delta \wedge \eta)^* &= \delta^* \wedge \eta^* \\ (\forall x : T \cdot \eta)^* &= \forall x : T^* \cdot \eta^* \\ (\forall x \in b \cdot \eta)^* &= \forall x \in b \cdot \eta^* \\ (\forall X \cdot \eta)^* &= \forall X \cdot \eta^* .\end{aligned}$$

All types are translated to themselves. This completes the translation. Part (2) is immediate given the translation. Part (1) follows by induction on the derivations in **CST^R**. All the rules, including the new axiom, are easy to verify. ■

We shall employ the whole translation again in connection with function specifications. For this reason we have provided the explicit details of the translation.

3.3 Comparing schema

For theoretical purposes, we shall need to compare schema specifications. There are two important relationships that we shall employ in the sequel. The first is the obvious one.

DEFINITION 3.15

Let

$$\begin{aligned}R[X_1, \dots, X_n] &\triangleq [x_1 : A_1, \dots, x_k : A_k \mid \eta] \\ S[X_1, \dots, X_n] &\triangleq [x_1 : A_1, \dots, x_k : A_k \mid \delta]\end{aligned}$$

be two schema specifications. We shall say they are **equivalent** (written $R \cong S$) iff

$$\forall X_1 \dots \forall X_n \cdot \forall x_1 : A_1 \dots \forall x_k : A_k \cdot \\ R[X_1, \dots, X_n](x_1, \dots, x_k) \leftrightarrow S[X_1, \dots, X_n](x_1, \dots, x_k)$$

EXAMPLE 3.16

The schema specification of union is equivalent to the schema with the predicate

$$(\forall x \in w \cdot x \in u \vee x \in v) \wedge (\forall x \in u \cdot x \in w) \wedge (\forall x \in v \cdot x \in w).$$

Although this notion will do a great deal of work for us in comparing schema, it will sometimes be too constraining and so we introduce another standard notion that generalizes it.

DEFINITION 3.17

Let

$$\begin{aligned}R[X_1, \dots, X_n] &\triangleq [x : I, y : O \mid \psi] \\ S[X_1, \dots, X_n] &\triangleq [x : I, y : O \mid \eta].\end{aligned}$$

We shall say that S is a **refinement** of R , written as

$$R \sqsubseteq S$$

iff

1. $\forall X_1 \cdot \dots \cdot \forall X_n \cdot \forall x : I \cdot \text{Dom}_R[X_1, \dots, X_n](x) \rightarrow \text{Dom}_S[X_1, \dots, X_n](x)$
2. $\forall X_1 \cdot \dots \cdot \forall X_n \cdot \forall x : I \cdot \text{Dom}_R[X_1, \dots, X_n](x) \rightarrow \forall y : O \cdot \psi[x, y] \leftrightarrow \eta[x, y]$.

We shall say that the two schema are **weakly equivalent** iff they refine each other, i.e. we write

$$R \approx S \triangleq R \sqsubseteq S \wedge S \sqsubseteq R.$$

Clearly if two schema are equivalent, they are weakly equivalent, i.e.

$$R \cong S \rightarrow R \approx S.$$

We shall unpack the connection in the other direction shortly. These notions will aid us in our investigation of many aspects of specification, including the following.

3.4 Total operations

There are some important special cases of schema which are theoretically and practically significant. We begin with *totality*.

DEFINITION 3.18

Let

$$R[X_1, \dots, X_n] \triangleq [x : I, y : O \mid \psi].$$

We shall say that R defines a **total** relation if

$$\forall X_1 \cdot \dots \cdot \forall X_n \cdot \forall x : I \cdot \text{Dom}_R[X_1, \dots, X_n](x). \quad \text{(TOT)}$$

We shall say that a many place operation is **total** iff

$$R[X_1, \dots, X_n] \triangleq [x : I_1 \otimes \dots \otimes I_k, y : O_1 \otimes \dots \otimes O_m \mid \psi[x_1, \dots, x_k, y_1, \dots, y_m]]$$

is.

Many of our operators are total.

PROPOSITION 3.19

pair, union, genunion, separation, dom and *ran* are total.

PROOF. We shall not prove all of these but rather illustrate the technique with simple union. For union, fix $u : \text{Set}(X)$. We use induction with the wff

$$\psi[v] = \exists w : \text{Set}(X) \cdot \forall x : X \cdot x \in w \leftrightarrow x \in u \vee x \in v.$$

If v is the empty set then we put $w = u$. Suppose that v has the form $x' \otimes y$. Assume inductively $\psi[y]$. Let w' be the guaranteed set. Then we put the required set for v to be $x' \otimes w'$. ■

Observe that for R and S total we have that weak equivalence implies equivalence, i.e.

$$R \approx S \rightarrow R \cong S.$$

By way of further unpacking these notions, also notice that:

PROPOSITION 3.20

If $R \sqsubseteq S$ and R is total then so is S .

Consequently, if one of the relations is total and they are weakly equivalent then they are equivalent. But obviously not all relations are total. However, given a relation, we can always define one which is total.

DEFINITION 3.21

Let

$$R[X_1, \dots, X_n] \triangleq [x : I, y : O \mid \psi].$$

Define the **totalization of R** as

$$R^T[X_1, \dots, X_n] \triangleq [x : I, y : O \mid \text{Dom}_R[X_1, \dots, X_n] \rightarrow \psi].$$

The reader might wish to compare this idea with that of [32]. The two notions are related but it will take us too far afield to say exactly how.

EXAMPLE 3.22

Consider the following specification of the predecessor relation:

$$\text{Pred} \triangleq [x : N, y : N \mid x = y^+].$$

The domain of this relation is given as

$$\text{Dom}_{\text{Pred}} \triangleq [x : N \mid \exists y : N \cdot x = y^+].$$

Since,

$$\forall x : N \cdot x > 0 \leftrightarrow \exists y : N \cdot x = y^+$$

the totalization is equivalent to the schema

$$\text{Pred}^T \triangleq [x : N, y : N \mid x > 0 \rightarrow x = y^+].$$

Observe that R and R^T are equivalent just in case R is total. Moreover, any relation can be refined to a total one.

PROPOSITION 3.23

For any R , R^T is total and $R \sqsubseteq R^T$.

PROOF. Clearly R^T is total. Moreover its domain extends that of R but on the domain of R it agrees with R . ■

3.5 Type independence

This brings us to the second general property of schema. The alert reader will have noticed that there is a difference between the basic relations of the theory (i.e. \in) and the present style of relation specification where new relation symbols take type arguments. This leads to the following idea.

DEFINITION 3.24

Let

$$R[X_1, \dots, X_n] \triangleq [x_1 : A_1, \dots, x_k : A_k \mid \psi]$$

be any schema specification. We shall say that R is **type independent** just in case R is equivalent to a schema of the form

$$S[X_1, \dots, X_n] \triangleq [x_1 : A_1, \dots, x_k : A_k \mid \eta]$$

where η contains no free type variables.

We then have:

THEOREM 3.25

For any type independent specification we may conservatively add a new relation symbol R^I which satisfies

$$\forall x_1 : A_1 \cdot \dots \cdot \forall x_k : A_k \cdot R^I(x_1, \dots, x_k) \leftrightarrow R[X_1, \dots, X_n](x_1, \dots, x_k) .$$

PROOF. We illustrate with the simple case.

$$\forall X \cdot \forall x : A[X] \cdot \forall y : B[X] \cdot R[X](x, y) \leftrightarrow \psi[X, x, y] .$$

Suppose that

$$\forall X \cdot \forall x : A[X] \cdot \forall y : B[X] \cdot \psi[X, x, y] \leftrightarrow \eta[x, y] .$$

We then we compile away in a similar manner to the explicit case but where the implicit relation is now interpreted as

$$R^I(a, b)^* = \eta(a, b) .$$

This clearly satisfies the condition. ■

PROPOSITION 3.26

Subset, extensional equivalence for sets, union, generalized union, separation, power and map are type independent.

PROOF. The first two we have met before and are immediate. We illustrate the rest with union and power set. For union, we know that the predicate is equivalent to

$$u \subseteq w \wedge v \subseteq w \wedge \forall x \in w \cdot x \in u \vee x \in v .$$

To show that the power set is type independent, we show that the predicate is equivalent to

$$\emptyset \in y \wedge (\forall z \in x \cdot \forall u \in y \cdot z \otimes u \in y) \wedge \forall z \in y \cdot z \subseteq x . \quad (*)$$

Assume $x : Set(X)$, $y : Set(Set(X))$. First assume *. Then

$$\forall w : Set(X) \cdot w \in y \rightarrow w \subseteq x$$

is automatic. So let $w : Set(X) \wedge w \subseteq x$. We argue by induction with the wff

$$\psi[w] \triangleq w \subseteq x \rightarrow w \in y .$$

If w is the empty set then we are done by *. If w has the form $x' \otimes y'$ then, by induction, we may assume $\psi[y']$ and we are done by *. Conversely, assume

$$\forall w : Set(X) \cdot w \in y \leftrightarrow w \subseteq x .$$

We have to show that $\emptyset \in y \wedge \forall z \in x \cdot \forall u \in y \cdot z \otimes u \in y$. The first conjunct is immediate. Assume that $z \in x \wedge u \in y$. So $z \in x \wedge u \subseteq x$. Hence, $z \otimes u \subseteq w$ and so $z \otimes u \in y$. ■

Where we can establish type independence we shall use the same name for the relation in its implicit manifestation. In particular, this enables us to circumvent the irritating build-up of type information in the predicates of specifications where the type information is already in the declaration.

For example, we may now specify power sets as

$$\begin{array}{|l} \text{Pow}[X] \\ \hline u : \text{Set}(X), v : \text{Set}(\text{Set}(X)) \\ \hline \forall x : \text{Set}(X) \cdot x \in v \leftrightarrow x \subseteq u \end{array}$$

It might be useful to develop some simple criteria for type independence. Many of these arise in connection with *schema calculus* [32]. But this will take us too far afield. We content ourselves with the following idea.

DEFINITION 3.27

Let

$$R[X_1, \dots, X_n] \triangleq [x : I, y : O \mid \psi]$$

be any schema specification. We shall say that R is **closed** if

$$\forall X_1 \dots \forall X_n \cdot \forall x : I \cdot \psi[x, y] \rightarrow y : O . \tag{CLO}$$

This is a natural condition on operations and simply demands that the type of the input determines the type of the output.

PROPOSITION 3.28

If R is type independent and satisfies closure then Dom_R and Ran_R are type independent.

PROOF. Let R be equivalent to

$$S[X] \triangleq [x : I, y : O \mid \eta]$$

where η contains no free type variables. Then Dom_R is equivalent to

$$[x : I \mid \exists y : O \cdot \eta] .$$

Given closure, this is equivalent to

$$[x : I \mid \exists Y \cdot \exists y : O[Y/X] \cdot \eta] .$$

■

This completes our introduction to relation specification. There is much more to say and more important examples to study but we have done enough to move on to *function specifications*.

4 Function specification

This is substantially different from the introduction of new relations: whereas the latter enrich the class of atomic wff, new function symbols enrich the class of individual terms and, in particular, new functions return values that can themselves be passed as arguments to other functions and relations. This will bring us closer to the specification style of **VDM** [7, 14]. Mathematically, this is a more subtle extension and *legitimacy* will be a more delicate matter. However, the addition of new functions is mathematically essential for the development of a useful theory of *numbers, sets and Cartesian products*.

4.1 Function application

For simplicity of notation, we shall employ the binary case to illustrate matters. However, as we shall see, given Cartesian products, one can easily extrapolate. To begin with we require the following notion.

DEFINITION 4.1

Let

$$R[X_1, \dots, X_n] \triangleq [x : I, y : O \mid \psi]$$

then we shall say that R is **functional** iff

$$\forall X_1 \cdot \dots \cdot \forall X_n \cdot \forall x : I \cdot \exists^{\leq 1} y : O \cdot \psi. \quad \text{(PF)}$$

The following are all easy consequences of extensionality.

PROPOSITION 4.2

Pair, union, genuinion, separation, Cartesian product are all functional.

However, relation specifications which happen to be functional are still relational in the sense that they are introduced as new relational symbols in the theory. They have not been introduced as *genuine* function symbols which can be applied to arguments in the standard way. This is the content of the following.

DEFINITION 4.3

Let

$$R[X_1, \dots, X_n] \triangleq [x : I, y : O \mid \psi]$$

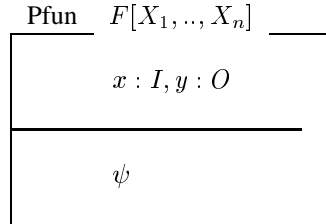
be functional. We may then introduce a new function symbol into the language via the following **function schema**

$$F[X_1, \dots, X_n] \triangleq_{Pfun} [x : I, y : O \mid \psi]. \quad \text{(FS)}$$

The specification is intended to introduce a new (partial) function symbol to the language and, in particular, for any A_1, \dots, A_n and t , $F[A_1, \dots, A_n](t)$ is a new term. In the special case where R is total we shall write

$$F[X_1, \dots, X_n] \triangleq_{Fun} [x : I, y : O \mid \psi].$$

The new function symbol does not occur in ψ , so at this point no recursion is intended. We shall deal with this in a later publication. We shall also write this specification in more graphic notation as



More generally, the specification of many place functions takes the form

$$F[X_1, \dots, X_n] \triangleq_{Pfun} [x_1? : I_1, \dots, x_k? : I_k, y! : O \mid \psi]$$

which is to be unpacked in terms of the product as the specification

$$F[X_1, \dots, X_n] \triangleq_{Pfun} [x? : I_1 \otimes \dots \otimes I_k, y! : O \mid \psi[x?_1/x_1?, \dots, x?_n/x_n?, y/y!]].$$

FS is intended to go beyond S. More specifically, given PF, FS is to be logically interpreted as the introduction of a new function symbol which satisfies the following axioms.

$$\forall X_1 \dots \forall X_n \cdot \forall x : I \cdot \text{Dom}_R[X_1, \dots, X_n](x) \rightarrow F[X_1, \dots, X_n](x) : O \quad (\mathbf{F}_1)$$

$$\forall X_1 \dots \forall X_n \cdot \forall x : I \cdot \text{Dom}_R[X_1, \dots, X_n](x) \rightarrow \psi[x, F[X_1, \dots, X_n](x)] . \quad (\mathbf{F}_2)$$

EXAMPLE 4.4

The following is a specification of the predecessor function

$$\text{Pred} \triangleq_{\text{Pfun}} [x : N, y : N \mid x = y^+].$$

This introduces predecessor as a new function symbol which satisfies

$$\begin{aligned} \forall x : N \cdot x > 0 &\rightarrow \text{Pred}(x) : N \\ \forall x : N \cdot x > 0 &\rightarrow \text{pred}(x)^+ = x . \end{aligned}$$

VDM uses the term *implementable* for the PF requirement (or rather the total version of it) but this seems inappropriate since it obviously does not guarantee that the function is in any sense *computable*. Instead, we continue to use the word *legitimate* but now to describe a function specification for which PF has been established.

Another way of looking at matters is instructive. Let

$$R[X_1, \dots, X_n] \triangleq [x : I, y : O \mid \psi]$$

be functional. Then we may introduce *application* for R via the axioms

$$\begin{aligned} \forall x : I \cdot \text{Dom}_R[X_1, \dots, X_n](x) &\rightarrow \text{App}_R[X_1, \dots, X_n](x) : O \\ \forall x : I \cdot \text{Dom}_R[X_1, \dots, X_n](x) &\rightarrow \psi[x, \text{App}_R[X_1, \dots, X_n](x)] . \end{aligned}$$

Of course, the two routes are formally identical and the only difference is the explicit declaration that a function is being defined. Our uniform use of schema notation for both highlights the relationship and difference.

EXAMPLE 4.5

The following is a functional specification of the Cartesian product operator on sets. It is total but it is non-trivial to show it.

<div style="display: flex; justify-content: space-between; align-items: center;"> Fun $\otimes[X]$ </div> <hr style="border: 0.5px solid black;"/> <div style="text-align: center; padding: 5px;"> $x? : \text{Set}(X), y? : \text{Set}(Y), z! : \text{Set}(X \otimes Y)$ </div> <hr style="border: 0.5px solid black;"/> <div style="text-align: center; padding: 5px;"> $\forall w : X \otimes Y \cdot w \in z \leftrightarrow w_1 \in x \wedge w_2 \in y$ </div>
--

EXAMPLE 4.6

The following is a specification of **application** for maps.

$$\begin{array}{l} \text{Pfun } \text{Mapp}[X, Y] \\ \hline z? : \text{Set}(X \otimes Y), u? \in X, w! : Y \\ \hline \text{Map}[X, Y](z) \wedge (u, w) \in z \end{array}$$

The following will prove useful shortly.

PROPOSITION 4.7

The following is a total function. It is also type independent.

$$\begin{array}{l} \text{Fun } \text{In}[X] \\ \hline u? : X, v? : \text{Set}(\text{Set}(X)), z! : \text{Set}(\text{Set}(X)) \\ \hline \forall x \in v \cdot u \otimes x \in z \\ \forall y \in z \cdot \exists x \in v \cdot y = u \otimes x \end{array}$$

PROOF. We first prove totality by induction on v with the wff

$$\exists z : \text{Set}(\text{Set}(X)) \cdot \forall x \in v \cdot u \otimes x \in z.$$

The case where v is empty we put $z = \{\{u\}\}$. So assume that v has the form $y \otimes w$. By induction,

$$\exists z' : \text{Set}(\text{Set}(X)) \cdot \forall x \in w \cdot u \otimes x \in z'.$$

The required set for $y \otimes w$ is then $(u \otimes y) \otimes z'$. Hence

$$\exists z : \text{Set}(\text{Set}(X)) \cdot \forall x \in v \cdot u \otimes x \in z.$$

Now given this set, the set required for the induction step is given by separation as

$$\{y \in z \cdot \exists x \in v \cdot y = u \otimes x\}.$$

Functionality follows from the extensional nature of sets. Independence is immediate. ■

We can now return to proof that the power-set constructor defines a total function.

PROPOSITION 4.8

Power is total, functional and type independent

PROOF. Type independence has already been established. For the other two, totality is the non-trivial part. We prove the result by induction where the induction wff is

$$\phi[x] = \exists y : \text{Set}(\text{Set}(X)) \cdot \forall z : \text{Set}(X) \cdot z \in y \leftrightarrow z \subseteq x.$$

If $x = \emptyset$ then the required set is \emptyset . If $x = u \otimes v$ then there are two cases. If $v = \emptyset$ then the required set is $\{\emptyset, \{u\}\}$. Otherwise, let v' be guaranteed by induction i.e. the power set of v . Now put the power set of $u \otimes v$ to be $\text{In}(u, v') \cup v'$. ■

EXAMPLE 4.9

The following provides a specification of a function that returns the domain of a set theoretic relation. We leave the investigation to the reader.

Fun	Dom
$u : \text{Set}(X \otimes Y), v : \text{Set}(X)$	
$\forall x : X \cdot x \in v \leftrightarrow \exists y : Y \cdot (x, y) \in u$	

The following provides a slightly different perspective on the introduction of new functions and has a more logical origin.

LEMMA 4.10

Given PF, in the language with a new function symbol added, F_1 and F_2 are equivalent to the following:

$$\forall X_1 \dots \forall X_n \cdot \forall x : I \cdot \forall y : O \cdot \text{Dom}_R(x) \rightarrow (F[X_1, \dots, X_n](x) = y \leftrightarrow \psi) . \quad (\mathbf{F})$$

PROOF. We illustrate with the case where $n = 1$. Assume F_1 and F_2 . Assume $x : I[X]$ and $y : O[X]$. Assume $\text{Dom}_R(x)$. Assume first $F[X](x) = y$. Then by F_2 , $\psi[x, F[X](x)]$. By F_1 , $F[X](x) : O[X]$. By E_2 , $\psi[x, y]$. On the other hand, given $\psi[x, y]$, and, by F_2 , $\psi[x, F[X](x)]$, functionality yields $y = F[X](x)$. Conversely, assume F_1 is immediate by definition of type membership in **CST** and the equivalence given by **F**. Moreover, given F_1 , F_2 is immediate: put $y = F[X](x)$ in **F**. ■

We can now see why PF is necessary. If we introduce a new function symbol without it, the theory may be rendered inconsistent. To see this, suppose that $x : I[X]$, $y : O[X]$, $y' : O[X]$ and

$$\psi[x, y] \wedge \psi[x, y'] \wedge y \neq y',$$

i.e. it is not *single-valued*. Then by **F**,

$$F[X](x) = y \wedge F[X](x) = y' \wedge y \neq y'.$$

So we cannot just specify a function via PS and stop; we must establish PF to be sure that the theory remains consistent. Finally note that ours is a very different approach to that adopted by **VDM** [13] which employs a 3-valued logic to deal with partial functions.

We shall refer to the above style of function definition as *indirect* since functions are being characterized logically rather than by indicating how to compute them. **VDM** uses the term *Implicit* but we have already adopted this description for polymorphism. In fact, **VDM** does not support the definition of *indirect polymorphic* functions in this very general form; it only allows such functions to be *directly* specified. Although this is actually a special case of the Indirect style, it is important enough to consider separately.

DEFINITION 4.11

Let I, O be any type terms and t any term, which may now contain free type variables. We assume that X_1, \dots, X_n include and exhaust all the type variables of I, O, t . We may then introduce a new

function symbol **directly** by the simple style of function specification

$$\begin{array}{c} \text{Fun } F[X_1, \dots, X_n] \\ \hline x : I, y : O \\ \hline y = t[x] \end{array}$$

Notice that this will be a total function just in case

$$\forall x : I \cdot t[x] : O.$$

EXAMPLE 4.12

Pairing provides a simple example where there are no type variables in the predicate.

$$\text{Pair}[X] \triangleq_{\text{Fun}} [x? : X, y? : X, z! : \text{Set}(X) \mid z = x \otimes y \otimes \emptyset].$$

Direct function definitions are common in mathematics and computer science. Indeed, much of the infrastructure of the theory will be constructed by a judicious mix of both indirect and direct ones. Typically, one presents an indirect definition followed by a sequence of direct ones supported by it.

4.2 Conservative extensions

We now turn to showing that such additions are conservative. Suppose that we can prove PF in **CST**. Let **CST^F** be the theory in which all the rules of **CST** are extended to this new language together with the axioms F_1, F_2 . Then we have:

THEOREM 4.13

CST^F is a conservative extension of **CST**.

This follows from the following.

LEMMA 4.14

There is a translation $*$ from the language of **CST^F** to the language of **CST** such that

1. if $\Gamma \vdash_{\text{CST}^F} \phi$ then $\Gamma^* \vdash_{\text{CST}} \phi^*$
2. if ϕ is a wff of **CST** then $\phi^* = \phi$.

PROOF. We illustrate with the simple case where there is only one type variable and the relation is total. We proceed as follows. First, using De Morgan's laws, we push all the negations through to atomic assertions. The translation then proceeds as in the relational case for all the connectives and quantifiers. Atomic assertions and their negations which do not contain F are compiled as before. This leaves us to deal with the atomic assertions and their negations which do contain F . These are transformed using the following:

$$\begin{aligned} \alpha[F[A](x)/y]^* &= x \in I[A] \wedge \exists u : O[A] \cdot \psi[A, x, u] \wedge \alpha[u/y] \\ (\neg\alpha[F[A](x)(x)/y])^* &= x \in I[A] \wedge \exists u : O[A] \cdot \psi[A, x, u] \wedge \neg\alpha[u/y]. \end{aligned}$$

Part (2) is immediate from the definition of the translation. Part (1) follows by induction on the derivations. Almost all the axioms and rules are automatic as they are in the relational case. This leaves us to check F_1 and F_2 . The former unpacks to

$$\forall x : I[A] \cdot \exists u : O[A] \cdot \exists v : O[A] \cdot \psi[A, x, u] \wedge u = v$$

which is true. For the latter we establish

$$\forall x \in I[X] \cdot (\exists u : O[X] \cdot \psi[X, x, u]) \rightarrow \psi[X, x, F[X](x)]$$

since, given totality, this immediately yields the result. We achieve this by induction on ψ . We may assume that ψ is in normal form. Suppose that ψ is an atomic wff. Then F_2 unpacks to the true:

$$\forall x : I[X] \cdot \exists u : O[X] \cdot \psi[X, x, u] \rightarrow \exists u : O[X] \cdot \psi[X, x, u].$$

The negative case is similar. Given that ψ is in normal form, all the induction cases are easy to check. For example, by induction

$$\begin{aligned} \forall x \in I[X] \cdot (\exists u : O[X] \cdot \eta[X, x, u]) &\rightarrow \eta[X, x, F[X](x)] \\ \forall x \in I[X] \cdot (\exists u : O[X] \cdot \delta[X, x, u]) &\rightarrow \delta[X, x, F[X](x)]. \end{aligned}$$

Hence,

$$\forall x \in I[X] \cdot (\exists u : O[X] \cdot \eta[X, x, u] \vee \delta[X, x, u]) \rightarrow \eta[X, x, F[X](x)] \vee \delta[X, x, F[X](x)].$$

This concludes part(2). ■

There has been a considerable amount of mathematical activity, centered upon the conservativeness properties of various kinds of operational extensions to logical theories. However, we cannot carry out a detailed analysis of all this work. Instead, we briefly examine the work that, either has most influenced us or has application to the development of **CST**.

[17] was one of the first to stress the importance of conservativeness in specification and this book influenced our early thinking about the subject. It was also one of the first works to stand back from actual languages and examine the theoretical foundations of specification.

In his attempt to base Z-style specification on a form of **HFST**, Hodges insisted on the conservativeness of specifications. Our approach owes much to this: it is a generalization of it to allow types to play a central role. Once they are in play, the issue of polymorphism comes to the fore — and our account has emphasized this.

[30] considers generalizations to the condition **TF** for the *safe* introduction of functions. These involve extracting functions from relations, where the graph of the function is either contained in or extends the relation. For non-generic functions, this generalizes our condition **PF**. It would be interesting to extend the current treatment to permit these generalizations. This would result in *upper* and *lower* functional specifications: extracting the containing and contained functions, respectively.

4.3 Functions with pre-conditions

VDM allows the specification of functions with *pre-conditions* and these are also considered in [30]. In this section we develop our style of functional specification to permit them. We first establish the following.

LEMMA 4.15

Let ϕ, ψ be any wff and I, O any type terms. We assume that X_1, \dots, X_n include and exhaust all the type variables of I, O, ϕ, ψ . Furthermore, we assume that x, y are the free individual variables of ψ and x is the only free individual variable of ϕ . Suppose that

$$\forall X_1 \cdot \dots \cdot \forall X_n \cdot \forall x : I \cdot \phi[x] \rightarrow \exists ! y : O \cdot \psi[x, y]. \quad (\text{PPF})$$

We may then legitimately introduce a new function symbol into the language that satisfies

$$\forall X_1 \cdot \dots \cdot \forall X_n \cdot \forall x : I \cdot \phi[x] \rightarrow F[X_1, \dots, X_n](x) : O \quad (\text{PREF}_1)$$

$$\forall X_1 \cdot \dots \cdot \forall X_n \cdot \forall x : I \cdot \phi[x] \rightarrow \psi[x, F[X_1, \dots, X_n](x)]. \quad (\text{PREF}_2)$$

PROOF. Given PPF, the following schema satisfies the above

$$F[X_1, \dots, X_n] \triangleq_{\text{Pfun}} [x : I, y : O \mid \psi].$$

■

DEFINITION 4.16

We shall write

$$F[X_1, \dots, X_n] \triangleq_{\text{Fun}} [x : I, y : O \mid \phi; \psi]$$

for a specification that is to be logically unpacked as the introduction of a new function symbol which satisfies $\text{PREF}_1, \text{PREF}_2$

The above shows that, where PPF is provable, such specifications are legitimate.

EXAMPLE 4.17

The following provides a specification, with pre-conditions, of map application.

$$\begin{array}{c} \text{Fun } \text{Mapp}[X, Y] \\ \hline z? : \text{Set}(X \otimes Y), u? \in X, w! : Y \\ \hline \text{Map}(z) \wedge u \in \text{Dom}[X, Y](z); \\ (u, w) \in z \end{array}$$

This provides us with a very general mechanism for the legitimate introduction of new function symbols.

4.4 *Type independence*

In parallel with relation specifications we may also drop the type variables in function specifications, but now we need not only functionality but also type independence.

THEOREM 4.18

If R is functional and type independent then we conservatively introduce a new function symbol F given axiomatically by

$$\forall X_1 \cdot \dots \cdot \forall X_n \cdot \forall x : I \cdot \text{Dom}_R[X_1, \dots, X_n](x) \rightarrow F(x) : O \quad (\text{IF}_1)$$

$$\forall X_1 \cdot \dots \cdot \forall X_n \cdot \forall x : I \cdot \text{Dom}_R[X_1, \dots, X_n](x) \rightarrow \psi[x, F(x)]. \quad (\text{IF}_2)$$

PROOF. We mimic the style of proof for explicit polymorphism. We illustrate with the following total case. We shall assume that ψ contains no free type variables-given type independence we can always reduce matters to such wff. We proceed as in the explicit case but translate

$$\begin{aligned} \alpha[F(x)/y]^* &= \exists X \cdot x \in I[X] \wedge \exists u : O[X] \cdot \psi[x, u] \wedge \alpha[u/y] \\ (\neg\alpha[F(x)(x)/y])^* &= \exists X \cdot x \in I[X] \wedge \exists u : O[X] \cdot \psi[x, u] \wedge \neg\alpha[u/y] . \end{aligned}$$

It is easy to check that IF₁ is satisfied. For IF₂, we establish that the following is sound under the translation.

$$\forall x \in I[X] \cdot (\exists u : O[X] \cdot \psi[x, u] \rightarrow \psi[x, F(x)]) .$$

Given totality, this immediately yields the result. We achieve this by induction on ψ . The proof then parallels the original. ■

Notice that as an upshot of this, we could take many of our relations as new implicitly polymorphic functions. For example, genunion would take the form of a new function symbols which satisfies

$$\begin{aligned} \forall x : Set(Set(X)) \cdot \cup(x) : Set(X) \\ \forall x : Set(Set(X)) \cdot \forall y : X \cdot y \in \cup(x) \leftrightarrow \exists z \in x \cdot y \in z \end{aligned}$$

A theory with pair, genunion, powerset and separation is a typed version of Zermelo set theory but with numbers forming a type and not a set. It is also a sub-theory of both Z and VDM.

4.5 States and operations

We complete our study of specifications by providing some example operations that are state based. Although simple, they should convince the reader that we can do standard Z-style specifications in the theory. The following is a specification of the state of a system which has two components one which is a set of items and the other which is a relation. The constraint or invariant insists that the domain of the relation and the set do not intersect.

EXAMPLE 4.19 (State)

$$\begin{array}{|l} \text{State}[X, Y] \\ \hline z : Set(X), v : Set(X \otimes Y) \\ \hline \text{Map}(v) \\ z \cap \text{Dom}(v) = \emptyset \end{array}$$

A concrete instance is a library data base where X represents the set of library items and Y the set of library users. Under this interpretation, $z : Set(X)$ represents the books currently on the shelves and $w : Set(X \otimes Y)$ that, for obvious reasons, is taken to be a map, provides the information of which books are on loan to which library users.

The following is a specification of an operation which updates our abstract state. In the library instance, this represents the operation of loaning a new item to a specified reader.

EXAMPLE 4.20 (Loan)

$Loan[X, Y]$
$u? : Set(X \otimes Y), v! : Set(X \otimes Y)$ $z? : Set(X), w! : Set(X)$ $x? : X, y? : Y$
$Map(u)$ $Map(v)$ $v = (x, y) \otimes u$ $w! = z - x$

This is a bit messy. Indeed, this kind of mess is a motivation for the development of the schema calculus — that proceeds in the present theory, as it does in \mathbf{Z} .

These examples represent a typical use made of sets and products in \mathbf{Z} for describing abstract states and their operations. However, this is all done in our simple set theory — not the full blown version of standard set theory. Indeed, we claim that when we write such \mathbf{Z} -style specifications, it is this simple notion of set that underlies our intuitions — not the infinite sets of standard set theory.

5 A type inference system

Type membership is embedded in the actual theory \mathbf{CST} and, consequently, types play an essentially mathematical role: they carve up the universe of objects into different kinds and our definitions and proofs are subject to these classifications. In this role they serve much the same function as they do in mathematics: they provide conceptual organization to the theory and its application. However, in computer science, types also have a *grammatical* function. This aspect is glossed over in \mathbf{CST} . To see what this function amounts to consider the following simple wff.

- (a) $x : N \wedge \exists y : N \cdot x \in y$
- (b) $x : N \wedge \exists y : Set(N) \cdot x = y$
- (c) $x : N \wedge \exists y : N \cdot x \leq y$.

Despite the fact that in the language of \mathbf{CST} they are all syntactically legitimate, there is a difference between them: the last is *well-typed* whereas the first two are not. Presumably, for them to be so, in the first y should be of type $Set(N)$ and in the second of type N .

5.1 The type inference system T

To deal with this, specification languages come equipped with some associated *type-checking* tools. Most often such a tool is supplied by a hidden program. However, in some cases [26, 5] it takes the more explicit form of a *type inference system*; much like those of the Lambda calculus [2]. However, these systems are rarely properly formulated as logical systems and, as a consequence, even their elementary properties are not documented. In this paper we develop and explore such a system for \mathbf{CST} . The investigation should serve as a template for the formulation and exploration of such systems.

The objective of the type inference system is to provide rules which determine whether wff, and ultimately *specifications*, are *well-typed*. The system (**T**) has two grammatical judgements.

$$\begin{array}{l} t : T \\ \phi \text{ prop} \end{array}$$

The first states that the term t has type T and the second that a wff is *well-typed*. We shall use Θ for a judgement of either of these two forms. Judgements are made relative to a declaration c which is here understood as a (possibly empty) set of assumptions

$$x_1 : T_1, \dots, x_m : T_m$$

where no individual variable is assigned more than one type. We write $\alpha(c)$ for the set of declared individual variables of a declaration c . We write $c \subseteq c'$ to indicate that c' is a consistent super-set of c and, where V is a set of variables, we shall write $c \upharpoonright V$ to denote c restricted to the variables in V . We shall say that a context c *covers* an expression e if $FV(e) \subseteq \alpha(c)$. Finally, we shall write $c, x : T$ for the context c updated with the assignment $x : T$, i.e. if $x \notin \alpha(c)$ then $x : T$ is added, whereas, if $x \in \alpha(c)$, its type is replaced by T .

The system is determined by the following axioms and rules. We shall suppress the declaration context unless the rule effects it.

There are two *structural rules*: *assumption* and *weakening*.

$$\mathbf{Ax} \quad x : T \vdash x : T \qquad \mathbf{W} \quad \frac{c \vdash \Theta}{c, x : T \vdash \Theta} \quad x \notin \alpha(c) \cup FV(\Theta).$$

For each built-in relation symbol, we associate a sequent that determines the conditions under which it is a proposition.

$$\mathbf{A}_1 \quad x : T, y : Set(T) \vdash x \in y \text{ prop}$$

$$\mathbf{A}_2 \quad x : T, y : T \vdash x = y \text{ prop}$$

$$\mathbf{A}_3 \quad x : N, y : N \vdash x < y \text{ prop} .$$

For each of the logical connectives there is a rule of formation that lays out the conditions under which complex wff are *well-typed*.

$$\begin{array}{lll} \mathbf{T}_1 \quad \Omega \text{ prop} & \mathbf{T}_2 \quad \frac{\phi \text{ prop}}{\neg \phi \text{ prop}} & \mathbf{T}_3 \quad \frac{\phi \text{ prop} \quad \psi \text{ prop}}{\phi \rightarrow \psi \text{ prop}} \\ \mathbf{T}_4 \quad \frac{\phi \text{ prop} \quad \psi \text{ prop}}{\phi \vee \psi \text{ prop}} & & \mathbf{T}_5 \quad \frac{\phi \text{ prop} \quad \psi \text{ prop}}{\phi \wedge \psi \text{ prop}} \\ \mathbf{T}_6 \quad \frac{c, x : T \vdash \phi \text{ prop}}{c \vdash \forall x : T \cdot \phi \text{ prop}} & & \mathbf{T}_7 \quad \frac{c, x : T \vdash \phi \text{ prop}}{c \vdash \exists x : T \cdot \phi \text{ prop}} \\ \mathbf{T}_8 \quad \frac{c \vdash \phi[X] \text{ prop}}{c \vdash \forall X \cdot \phi \text{ prop}} & & \mathbf{T}_9 \quad \frac{c \vdash \phi[X] \text{ prop}}{c \vdash \exists X \cdot \phi \text{ prop}} \end{array}$$

In the rules T_8 and T_9 , $X \notin FTV(c)$. The rules for the bound quantifiers are given as follows. Notice that the premiss requires that the wff be a proposition in the appropriate context.

$$\mathbf{T}_{10} \frac{c, x : N \vdash \phi \text{ prop} \quad c \vdash n : N}{c \vdash \forall x < n \cdot \phi \text{ prop}} \quad \mathbf{T}_{11} \frac{c, x : N \vdash \phi \text{ prop} \quad c \vdash n : N}{c \vdash \exists x < n \cdot \phi \text{ prop}}$$

$$\mathbf{T}_{12} \frac{c, x : T \vdash \phi \text{ prop} \quad c \vdash s : Set(T)}{c \vdash \forall x \in s \cdot \phi \text{ prop}} \quad \mathbf{T}_{13} \frac{c, x : T \vdash \phi \text{ prop} \quad c \vdash s : Set(T)}{c \vdash \exists x \in s \cdot \phi \text{ prop}}$$

Almost finally, we provide the rules for the judgement $t : T$. These are quite obvious and should not cause one to pause since they are the closure conditions for our types. Indeed, we give them the same names as the corresponding axioms of **CST**.

$$\begin{aligned} \mathbf{N}_1 & 0 : N \\ \mathbf{N}_2 & x : N \vdash x^+ : N \\ \mathbf{S}_1 & \emptyset : Set(X) \\ \mathbf{S}_2 & x : X, y : Set(X) \vdash x \otimes y : Set(X) \\ \mathbf{P}_1 & x : X, y : Y \vdash (x, y) : X \otimes Y \\ \mathbf{P}_2 & z : X \otimes Y \vdash \pi_1(z) : X \\ \mathbf{P}_3 & z : X \otimes Y \vdash \pi_2(z) : Y \end{aligned}$$

Finally, since we have no *logical* quantification rules we require substitution rules.

$$\mathbf{Sub} \frac{c, x : T \vdash \Theta \quad c \vdash t : T}{c \vdash \Theta[t/x]} \quad \mathbf{SUB} \frac{c[X] \vdash \Theta[X]}{c[T/X] \vdash \Theta[T/X]}$$

This completes the rules of the system **T**. It should be clear enough how we use it but we provide a simple example anyway.

EXAMPLE 5.1

Given the axiom A_1 for membership and the conjunction rule (T_5) we have:

$$x : X, y : X, z : Set(X) \vdash x \in z \wedge y \in z \text{ prop.}$$

By the axiom A_2 for equality:

$$x : X, y : X, u : X \vdash u = x \vee u = y \text{ prop.}$$

By the rule for set quantification

$$x : X, y : X, z : Set(X) \vdash \forall u \in z \cdot u = x \vee u = y \text{ prop.}$$

Finally, by the conjunction rule

$$x : X, y : X, z : Set(X) \vdash x \in z \wedge y \in z \wedge \forall u \in z \cdot u = x \vee u = y \text{ prop.}$$

There are some immediate properties of the system which parallel the standard results for any well behaved system of type inference. The following parallel the results for the type systems for the lambda calculus [2].

PROPOSITION 5.2
In the system **T**

1. If $c \vdash \Theta$ then c covers Θ .
2. If $c \vdash \Theta$ then $c \upharpoonright FV(\Theta) \vdash \Theta$.
3. If $c \vdash \Theta$ and $c \subseteq c'$ then $c' \vdash \Theta$.

PROOF. All are by induction on the derivations. We illustrate with some characteristic cases. Consider part (1). The structural rules are automatic: given that c covers the premise it covers the conclusion. Next consider the type quantifier rule.

$$\frac{c, x : T \vdash \phi \text{ prop}}{c \vdash \forall x : T \cdot \phi \text{ prop}}$$

If $c, x : T$ covers the premise then c covers the conclusion. For part (2) we again illustrate with the quantifier rule. By induction,

$$(c, x : T) \upharpoonright FV(\phi) \vdash \phi \text{ prop.}$$

Hence, regardless of whether $x \in FV(\phi)$, we have:

$$c \upharpoonright FV(\forall x : T \cdot \phi), x : T \vdash \phi \text{ prop.}$$

Hence, by the rule,

$$c \upharpoonright FV(\forall x : T \cdot \phi) \vdash \forall x : T \cdot \phi \text{ prop.}$$

For part (3) we again illustrate with the quantifier rule. Suppose

$$c \vdash \forall x : T \cdot \phi \text{ prop}$$

follows from $c, x : T \vdash \phi \text{ prop}$. By induction, for $c' \supseteq c$, $c', x : T \vdash \phi \text{ prop}$. Hence, $c' \vdash \forall x : T \cdot \phi \text{ prop}$. ■

The next is the standard generation lemma that provides the means of automatically checking that an expression is well-typed by using the system backwards.

PROPOSITION 5.3 (Generation)

1. If $c \vdash t_1 = t_2 \text{ prop}$ then $c \vdash t_1 : T$ for some type T . Moreover, $c \vdash t_1 : T$ iff $c \vdash t_2 : T$.
2. If $c \vdash t \in s \text{ prop}$ then $c \vdash t : T$ and $c \vdash s : \text{Set}(T)$ for some T .
3. If $c \vdash t < s \text{ prop}$ then $c \vdash t : N$ and $c \vdash s : N$.
4. If $c \vdash \neg \phi \text{ prop}$ then $c \vdash \phi \text{ prop}$.
5. If $c \vdash \phi \circ \psi \text{ prop}$ then $c \vdash \phi \text{ prop}$ and $c \vdash \psi \text{ prop}$ ($\circ = \vee, \wedge, \rightarrow$).
6. If $c \vdash Qx : T \cdot \phi \text{ prop}$ then $c, x : T \vdash \phi \text{ prop}$ ($Q = \exists, \forall$).
7. If $c \vdash Qx < t \cdot \phi \text{ prop}$ then $c \vdash t : N$ and $x : N \vdash \phi \text{ prop}$ ($Q = \exists, \forall$).
8. If $c \vdash Qx \in t \cdot \phi \text{ prop}$ then $c \vdash t : \text{Set}(T)$ for some T and $x : T \vdash \phi \text{ prop}$ ($Q = \exists, \forall$).
9. If $c \vdash QX \cdot \phi \text{ prop}$ then $c \vdash \phi[X] \text{ prop}$ ($Q = \exists, \forall$).
10. If $c \vdash t^+ : N$ then $c \vdash t : N$.
11. If $c \vdash (t_1, t_2) : T_1 \otimes T_2$ then $c \vdash t_1 : T_1$ and $c \vdash t_2 : T_2$.
12. If $c \vdash \pi_1(t) : T_1$ then $c \vdash t : T_1 \otimes T_2$ for some T_2 .

13. If $c \vdash a \otimes b : Set(T)$ then $c \vdash a : T$ and $c \vdash b : Set(T)$ for some T .

PROOF. By induction on the derivations. Each of the antecedents of the above conditionals can only be result of a rule application whose premisses are the consequents of the corresponding conditional or the result of a substitution or the application of a structural rule. The first group of cases are immediate. For the term substitution rules we illustrate with the first and the following instance. Suppose that

$$c \vdash a[t] = b[t] \text{ prop}$$

follows from sub. The premisses yield

$$c, x : T \vdash a[x] = b[x] \text{ prop} \quad c \vdash t : T.$$

Then by induction,

$$c, x : T \vdash a[x] : S$$

for some type S . We now employ the substitution rule itself. The structural rules are easy to verify for all the cases. The following derivation involving the negation instance of the first rule, illustrates the argument. Suppose that

$$c, x : T \vdash \neg\phi \text{ prop}$$

follows by the rule

$$\frac{c \vdash \neg\phi \text{ prop}}{c, x : T \vdash \neg\phi \text{ prop}}$$

Then by induction $c \vdash \phi \text{ prop}$ and so by the structural rule itself $c, x : T \vdash \phi \text{ prop}$. ■

We also have the following admissible rule.

PROPOSITION 5.4 (Back substitution)

If $c \vdash t : T$ and $c \vdash \Theta[t/x]$ then $c, x : T \vdash \Theta[x]$.

PROOF. By induction on the derivations. We illustrate with some significant cases. Suppose that

$$c \vdash a[t/x] \in b[t/x] \text{ prop}.$$

Then by the generation lemma, for some A ,

$$c \vdash a[t/x] : A \text{ and } c \vdash b[t/x] : Set(A).$$

By induction, for some type T , we have:

$$c, x : T \vdash a[x] : A \text{ and } c, x : T \vdash b[x] : Set(A) \text{ and } c \vdash t : T$$

so we are finished by the membership axiom. Next assume that

$$c \vdash (\phi \wedge \psi)[t/x] \text{ prop},$$

then

$$c \vdash \phi[t/x] \text{ prop and } c \vdash \psi[t/x] \text{ prop}.$$

Hence, by induction,

$$c, x : T \vdash \phi \text{ prop and } c, x : T \vdash \psi \text{ prop and } c \vdash t : T$$

and we are finished by the conjunction rule. Now consider part two. We again illustrate with the following case. Suppose that

$$c \vdash a[t/x] \in b[t/x] \text{ prop and } c \vdash t : T.$$

Then by the generation lemma, for some A ,

$$c \vdash a[t/x] : A \text{ and } c \vdash b[t/x] : \text{Set}(A).$$

By induction, we have:

$$c, x : T \vdash a[x] : A \text{ and } c, x : T \vdash b[x] : \text{Set}(A)$$

so we are finished by the membership axiom. ■

The following informs us that any judgement of the form $t : T$ that is provable in the type system, is also provable in the logic. This is the first step in charting the relationship between the two systems.

PROPOSITION 5.5

If $c \vdash_{\mathbf{T}} t : T$ then $c \vdash_{\mathbf{CST}} t : T$.

PROOF. By induction on the derivations in \mathbf{T} . Since, apart from the substitution rule, the rules of \mathbf{T} are also rules of \mathbf{CST} . Moreover, the substitution rules follow directly from the rules of the logic. Hence we are finished. ■

The converse is not provable. This marks a difference between the *grammatical* and *logical* roles of types. Logically, the type membership assertion forms part of the logic of the system. These two systems capture the exact difference between these two roles and pinpoints the fact that the properties of type membership emerge from the theory, i.e. \mathbf{T} versus \mathbf{CST} .

5.2 Specifications

The principal application of \mathbf{T} is to type-check specifications. To facilitate this we must introduce a rule for schema specifications that allows wff involving the new relation or function symbol to be checked.

Relation specifications are treated in a similar fashion to the other atomic wff but now added as a rule with the predicate of the specification supplying the premiss. Given a specification

$$R[X_1, \dots, X_m] \triangleq [x_1 : T_1, \dots, x_n : T_n \mid \phi]$$

we extend the system with the rule

$$\frac{x_1 : T_1, \dots, x_n : T_n \vdash \phi[X_1, \dots, X_m, x_1, \dots, x_n] \text{ prop}}{x_1 : T_1, \dots, x_n : T_n \vdash R[X_1, \dots, X_m](x_1, \dots, x_n) \text{ prop}} \quad (\mathbf{ST})$$

Call this system $\mathbf{T+ST}$. It is then easy to show, by induction on the derivations, that:

PROPOSITION 5.6

If $c \vdash_{\mathbf{T+ST}} \Theta$ then $c \vdash \Theta^*$ where Θ^* is obtained from Θ by replacing each occurrence of $R[X_1, \dots, X_m](x_1, \dots, x_n)$ by $\phi[X_1, \dots, X_m, x_1, \dots, x_n]$.

So, in the obvious sense, this addition to the type inference system is conservative. We can also easily establish the following.

PROPOSITION 5.7

The following rules are derivable

1. $x : X, y : X, z : Set(X) \vdash Pair_X(x, y, z) \text{ prop.}$
2. $x : Set(X), y : Set(X), z : Set(X) \vdash Union_X(x, y, z) \text{ prop.}$
3. $x : Set(Set(X)), y : Set(X) \vdash Genunion_X(x, y) \text{ prop.}$
4. $x : Set(X), y : Set(Set(X)) \vdash Pow_X(x, y) \text{ prop.}$
5.
$$\frac{y : Set(X) \vdash \phi[y] \text{ prop}}{x : Set(X) \vdash \{y \in x \cdot \phi\} : Set(X)}.$$

PROOF. We illustrate with the second. According to the new rule for the specification, it is sufficient to show that

$$x : Set(X), y : Set(X), z : Set(X) \vdash \forall u : X \cdot u \in x \leftrightarrow u \in y \vee u \in z \text{ prop}$$

and this is straightforward. ■

Much the same is true if we wish to type-check relations which are type independent. Here we add the rule

$$\frac{x_1 : T_1, \dots, x_n : T_n \vdash \phi[x_1, \dots, x_n] \text{ prop}}{x_1 : T_1, \dots, x_n : T_n \vdash R(x_1, \dots, x_n) \text{ prop}}$$

The conservative extension result follows the same route as the general case.

The situation with the addition of new function symbols is similar to the basic operations of the theory except that we need to ensure they are well-typed. Suppose that we have legitimately introduced a new function symbol via the specification

$$F[X] \triangleq_{Pfun} [x : I[X], y : O[X] \mid \psi[X, x, y]].$$

Then we introduce a new type inference rule

$$\frac{x : I[X], y : O[X] \vdash \psi[X, x, y] \text{ prop}}{x : I[X] \vdash F[X](x) : O[X] \text{ prop}} \quad FT$$

Call this system **T+FT**. Once more, this addition is conservative. In the following * is the translation from the system with a new function symbol to the system without.

PROPOSITION 5.8

If $c \vdash_{T+FT} \Theta$ then $c \vdash \Theta^*$ where Θ^* is obtained from Θ by replacing every wff by its translation.

PROOF. We illustrate with the total case; the partial one causes no additional complications. We proceed as in the original proof by putting all wff in normal form and then replace each occurrence of $F[X](x) : O[X]$ by $x : I[X] \wedge \exists y : O[X] \cdot \psi[X, x, y]$. We have then only to show that the following is derivable

$$\frac{x : I[X], y : O[X] \vdash \psi[X, x, y] \text{ prop}}{x : I[X] \vdash \exists y : O[X] \cdot \psi[X, x, y] \text{ prop}}$$

which is clear. ■

Observe that, given the definition of type membership, by the generation lemma, we have that the following is derivable.

$$\text{If } c \vdash t : T \text{ prop then } c \vdash t : T.$$

Hence, given FT, the following is admissible

$$\frac{x : I[X], y : O[X] \vdash \psi[X, x, y] \text{ prop}}{x : I[X] \vdash F[X](x) : O[X]}$$

It follows that the following are derivable

$$\frac{a : \text{Set}(\text{Set}(X))}{\cup_X a : \text{Set}(X)} \quad \frac{a : \text{Set}(X)}{\text{Set}_X(a) : \text{Set}(\text{Set}(X))}$$

Finally, we may drop the type variables on the function symbol when it has been shown to be type independent. For example, we then have the following derived rules for generalized union and power set.

$$\frac{a : \text{Set}(\text{Set}(X))}{\cup a : \text{Set}(X)} \quad \frac{a : \text{Set}(X)}{\text{Set}(a) : \text{Set}(\text{Set}(X))}$$

6 A strongly typed theory

Is **CST** a typed theory? Certainly it has types and objects in the theory have them. However, we have observed that not every wff provable in **CST** is well-typed. This follows from our preliminary discussion on the relationship between **T** and **CST**. Hence, **CST** is not a *typed theory* in the traditional sense of higher order logic (**HOL**) where only *well-typed* wff are provable. In this section we develop a version of **CST** (**CST_T**) that is typed in this more traditional sense. This will enable us to fully explore the relationship between **CST** and **T**. In particular, we shall show that **CST** is a conservative extension of **CST_T** — with respect to well-typed expressions. This will throw some mathematical light on the rather curious and somewhat murky relationship between *type* and *logical inference* that has emerged from the development of specification languages.

6.1 The theory **CST_T**

This is a marriage of **CST** and **T**. Consequently, the theory now has three judgements which combine those of **CST** and **T**.

$$\begin{array}{l} t : T \\ \phi \text{ prop} \\ \phi \end{array}$$

We use Θ for a judgement of any of the above kinds. In particular, and this is important, we now drop the following definition.

$$t : T \triangleq \exists x : T \cdot x = t.$$

Type membership is now taken as a primitive judgement in all the axioms and rules, i.e. it is no longer to be interpreted as the above wff.

Such judgements are made relative to a context Γ that now contains wff and type assignments of the form $x : T$. We shall write

$$\Gamma \vdash_{\text{CST}_T} \Theta$$

if the sequent follows from the following rules — where we drop the subscript. We shall write c_Γ for the subset of Γ which consists of its set of type assignments.

We begin with the structural rules which now take the following form. These subsume the structural rules of **CST** and **T**.

$$\begin{array}{ccc} \mathbf{Ax}_1 & x : T \vdash x : T & \mathbf{Ax}_2 \quad \frac{\Gamma \vdash \phi \text{ prop}}{\Gamma, \phi \vdash \phi} \\ \mathbf{W}_1 & \frac{\Gamma \vdash \Theta}{\Gamma, x : T \vdash \Theta} & \mathbf{W}_2 \quad \frac{\Gamma \vdash \Theta \quad \Gamma \vdash \phi \text{ prop}}{\Gamma, \phi \vdash \Theta} \\ & \text{where } x \notin FV(\Gamma) \cup FV(\Theta) & \end{array}$$

The equality axioms/rules remain intact except for the need to include a premiss in E_2 .

$$\mathbf{E}_1 \quad \forall x : X \cdot x = x \quad \mathbf{E}_2 \quad \frac{\Gamma, x : X \vdash \phi[x] \text{ prop}}{\Gamma \vdash \forall x : X \cdot \forall y : X \cdot x = y \rightarrow (\phi[x] \rightarrow \phi[y])}$$

The propositional logical rules are modified to preserve *propositions* from premisses to conclusions. However, only the following rules need to be modified.

$$\begin{array}{ccc} \mathbf{L}_2 & \frac{\Gamma \vdash \Omega \quad \Gamma \vdash \phi \text{ prop}}{\Gamma \vdash \phi} & \\ \mathbf{L}_8 & \frac{\Gamma \vdash \phi \quad \Gamma \vdash \psi \text{ prop}}{\Gamma \vdash \phi \vee \psi} & \mathbf{L}_9 \quad \frac{\Gamma \vdash \psi \quad \Gamma \vdash \phi \text{ prop}}{\Gamma \vdash \phi \vee \psi} \end{array}$$

The axioms and rules for the types remain exactly as before except for the following. We must state the induction axioms as rules to get preservation of *being a proposition* from premisses to conclusion.

$$\begin{array}{c} \mathbf{N}_5 \quad \frac{\phi[0] \wedge \forall x : N \cdot \phi[x] \rightarrow \phi[x^+]}{\forall x : N \cdot \phi[x]} \\ \mathbf{S}_5 \quad \frac{\phi[\emptyset] \wedge \forall x : X \cdot \forall y : Set(X) \cdot \phi[y] \rightarrow \phi[x \otimes y]}{\forall y : Set(X) \cdot \phi[y]} \end{array}$$

Almost finally, we modify the bounded quantifier rules to include type inference premisses.

$$\begin{array}{c} \mathbf{N}_8 \quad \frac{x : N \vdash \phi \text{ Prop}}{\forall y : N \cdot \forall x < y \cdot \phi \leftrightarrow \forall x : N \cdot x < y \rightarrow \phi} \\ \mathbf{N}_9 \quad \frac{x : N \vdash \phi \text{ Prop}}{\forall y : N \cdot \exists x < y \cdot \phi \leftrightarrow \exists x : N \cdot x < y \wedge \phi} \\ \mathbf{S}_8 \quad \frac{x : X \vdash \phi \text{ Prop}}{\forall y : Set(X) \cdot \forall x \in y \cdot \phi \leftrightarrow \forall x : X \cdot x \in y \rightarrow \phi} \\ \mathbf{S}_9 \quad \frac{x : X \vdash \phi \text{ Prop}}{\forall y : Set(X) \cdot \exists x \in y \cdot \phi \leftrightarrow \exists x : X \cdot x \in y \wedge \phi} \end{array}$$

Alternatively, these could also be stated as pairs of rules without the need to include the type premisses. Finally we admit the substitution rules

$$\mathbf{Sub} \quad \frac{\Gamma, x : T \vdash \Theta \quad \Gamma \vdash t : T}{\Gamma \vdash \Theta[t/x]} \quad \mathbf{SUB} \quad \frac{\Gamma[X] \vdash \Theta[X]}{\Gamma[T/X] \vdash \Theta[T/X]}$$

Of course we do not need these for the purely logical cases since they are already derivable from the quantifier rules. This completes the statement of the theory CST_T .

It is obviously more cumbersome to use CST_T than CST since a great deal of type inference has to be done in the process of carrying out the proofs. Although this can all be mechanized, it is harder to use by hand. However, CST_T does have the following very pleasant property that is not shared by CST .

THEOREM 6.1 (Strong typing)

- (1) If $\Gamma \vdash_{\text{CST}_T} \psi$ then for each ϕ in $\Gamma \cup \{\psi\}$, $c_\Gamma \vdash_{\mathbf{T}} \phi \text{ prop}$.
- (2) If $\Gamma \vdash_{\text{CST}_T} t : T$ then $c_\Gamma \vdash_{\mathbf{T}} t : T$.
- (3) If $\Gamma \vdash_{\text{CST}_T} \phi \text{ prop}$ then $c_\Gamma \vdash_{\mathbf{T}} \phi \text{ prop}$.

PROOF. By induction on the structure of derivations. For part (1) we first consider the inference rules. Most of the rules are easy to check; we provide some illustrations. Consider first the implication introduction rule. Consider the premiss

$$\Gamma, \phi \vdash \psi.$$

By part (1) and induction, $c_\Gamma \vdash_{\mathbf{T}} \phi \text{ prop}$ and $c_\Gamma \vdash_{\mathbf{T}} \psi \text{ prop}$. This yields $c_\Gamma \vdash_{\mathbf{T}} \phi \rightarrow \psi \text{ prop}$. Next consider disjunction introduction. The premiss is

$$\Gamma \vdash \phi \quad c_\Gamma \vdash_{\mathbf{T}} \psi \text{ prop}.$$

By induction, $c_\Gamma \vdash_{\mathbf{T}} \phi \text{ prop}$. The result now follows by the grammar rule for disjunction. Next consider existential quantification introduction.

$$\frac{\Gamma \vdash a : T \quad \Gamma \vdash \phi[a/x]}{\Gamma \vdash \exists x : T \cdot \phi}$$

By induction

$$c_\Gamma \vdash_{\mathbf{T}} a : T \quad c_\Gamma \vdash_{\mathbf{T}} \phi[a/x] \text{ prop}.$$

By the properties of the type system, we may assume that $x \notin \alpha(c_\Gamma)$. Hence, by the back substitution lemma:

$$c_\Gamma, x : T \vdash_{\mathbf{T}} \phi[x] \text{ prop}.$$

We are then done by the existential formation rule. The axioms and rules for the types are equally easy to check. In particular the axioms are well-typed by their declaration contexts. The only other concern is with the structural rules which are easy to check. This completes part (1). Parts (2) and (3) are by structural induction and are easy to check. ■

Part (1) ensures that only propositions are provable and guarantees that if a propositional term is used in an assumption, the assignment context will guarantee that it is provably a proposition. Notice that, via an obvious induction, the theorem also yields that all derivations are well-typed, i.e. all formulae in the derivation will be. Part (2) is exactly the property which distinguished the classifier and grammatical roles of types in the theory CST .

6.2 *CST*, *T* and *CST_T*

We shall show that, with respect to well-typed expressions, *CST* is a conservative extension of *CST_T*. This is achieved by, relative to a context *c*, translating *CST* into *CST_T* in such a way that each translated expression is, relative to *c*, *well-typed*.

The types are translated to themselves: we have only to translate the wff and terms. We deal first with the wff. The tricky clauses are those for the atomic wff and here we translate the *ill-typed* cases as false.

$$\begin{aligned} (t \in s)^c &= \left\{ \begin{array}{l} t^c \in s^c \text{ if } c \vdash_T t^c : T \text{ and } c \vdash_T s^c : \text{Set}(T) \\ \Omega \text{ otherwise} \end{array} \right\} \\ (t = s)^c &= \left\{ \begin{array}{l} t^c = s^c \text{ if } c \vdash_T t^c : T \text{ and } c \vdash_T s^c : T \text{ for some } T \\ \Omega \text{ otherwise} \end{array} \right\} \\ (t < s)^c &= \left\{ \begin{array}{l} t^c < s^c \text{ if } c \vdash_T t^c : N \text{ and } c \vdash_T s^c : N \\ \Omega \text{ otherwise} \end{array} \right\} \end{aligned}$$

For the main connectives and quantifiers, matters are straightforward.

$$\begin{aligned} (\phi \circ \psi)^c &= \phi^c \circ \psi^c \quad \circ \text{ any connective} \\ (Qx : T \cdot \phi)^c &= Qx : T \cdot \phi^{c,x:T} \quad Q = \exists, \forall \\ (QX \cdot \phi)^c &= QX \cdot \phi^{c,x:T} \quad Q = \exists, \forall. \end{aligned}$$

The translation of the bounded quantifiers follows a similar pattern where we have to take care of the bad cases explicitly.

$$\begin{aligned} (\exists x < b \cdot \phi)^c &= \left\{ \begin{array}{l} \exists x < b \cdot \phi^{c,x:T} \text{ if } c \vdash_T b^c : N \\ \Omega \text{ otherwise} \end{array} \right\} \\ (\forall x < b \cdot \phi)^c &= \left\{ \begin{array}{l} \forall x < b \cdot \phi^{c,x:T} \text{ if } c \vdash_T b^c : N \\ \Omega \text{ otherwise} \end{array} \right\} \\ (\exists x \in b \cdot \phi)^c &= \left\{ \begin{array}{l} \exists x \in b \cdot \phi^{c,x:T} \text{ if } c \vdash_T b^c : \text{Set}(T) \\ \Omega \text{ otherwise} \end{array} \right\} \\ (\forall x \in b \cdot \phi)^c &= \left\{ \begin{array}{l} \forall x \in b \cdot \phi^{c,x:T} \text{ if } c \vdash_T b^c : \text{Set}(T) \\ \Omega \text{ otherwise} \end{array} \right\} \end{aligned}$$

Finally, for the translation of the terms we have only to be careful about the cases where the types are inappropriate and then assign the result some arbitrary value.

$$\begin{aligned}
0^c &= 0 \\
(t^+)^c &= \begin{cases} (t^c)^+ & \text{if } c \vdash_T t^c : N \\ 0 & \text{otherwise} \end{cases} \\
(t_1, t_2)^c &= (t_1^c, t_2^c) \\
\pi_i(t)^c &= \begin{cases} \pi_i(t^c) & \text{if } c \vdash_T t^c : A \otimes B \text{ for some } A, B \\ 0 & \text{otherwise} \end{cases} \\
\Phi^c &= \Phi \\
(a \otimes b)^c &= \begin{cases} a^c \otimes b^c & \text{if } c \vdash_T a^c : T \text{ and } c \vdash_T b^c : \text{Set}(T) \\ \emptyset & \text{otherwise} \end{cases}
\end{aligned}$$

This completes the translation.

LEMMA 6.2 (Substitution lemma)

1. If $c \vdash_T t^c : T$ then $\phi[t/x]^c = \phi^{c, x:T}[t^c/x]$
2. If $c \vdash_T t^c : T$ then $s[t/x]^c = s^{c, x:T}[t^c/x]$

PROOF. By simultaneous induction on the terms and wff. ■

Our first significant result shows that the translation always yields well-typed expressions.

PROPOSITION 6.3 (Typing)

Let c be any context.

1. If c covers t/ϕ and $c \subseteq e$ then $t^c = t^e$ and $\phi^c = \phi^e$
2. If c covers ϕ then $c \vdash_T \phi^c \text{ prop}$
3. If c covers t then $c \vdash_T t^c : T$ for some T
4. If $c \vdash_T \phi^c \text{ prop}$ then $\phi = \phi^c$

PROOF. (1) is by induction on the terms and wff. For parts 2 and 3, we employ a simultaneous induction on the syntax of terms and wff. For example, consider universal quantification. Assume that c covers $\forall x : T \cdot \phi$. We may assume that $FV(\forall x : T \cdot \phi)$ are exactly the variables of c . By induction,

$$c, x : T \vdash_T \phi^{c, x:T} \text{ prop}.$$

Hence by the type rule for quantifiers

$$c \vdash_T \forall x : T \cdot \phi^{c, x:T} \text{ prop}.$$

Part (4) is also by induction but we only need inspection of the clauses. ■

We can now prove the main theorem which ensures the translation preserves provability in the two systems.

THEOREM 6.4 (Soundness)

If

$$\Gamma \vdash_{\text{CST}} \phi$$

and c covers Γ, ϕ then

$$\Gamma^c, c \vdash_{\text{CST}_T} \phi^c$$

where Γ^c is the translated context.

PROOF. By induction on the derivations in **CST**. Each of the axioms is easy to verify. We illustrate the logical rules with disjunction introduction, implication elimination and the universal quantification introduction rule. For the first assume c covers $\phi \vee \psi$. Then by induction

$$\Gamma^c, c \vdash \phi^c.$$

Hence, since $c \vdash \psi^c$ *prop*, we are finished by the **CST_T** disjunction rule. For implication elimination,

$$\frac{\Gamma \vdash \phi \quad \Gamma \vdash \phi \rightarrow \psi}{\Gamma \vdash \psi}.$$

Let c cover the conclusion and expand it to e to cover ϕ as follows: for each $y \in FV(\Gamma \cup \{\phi\}) - FV(\psi)$ we add $y : N$ to c . Then we have by induction

$$\Gamma^e, e \vdash \phi^e \quad \Gamma^e, e \vdash \phi^e \rightarrow \psi^e.$$

Hence by the rules of **CST_T**,

$$\Gamma^e, e \vdash \psi^e.$$

Now by the lemma on the properties of the translation,

$$\Gamma^c, e \vdash \psi^c.$$

For simplicity, suppose that y is the only variable in $FV(\Gamma \cup \{\phi\}) - FV(\psi)$. We then have:

$$\Gamma^c, c \vdash \forall y : N \cdot \psi^c.$$

Since $y \notin FV(\psi)$, this yields

$$\Gamma^c, c \vdash \psi^c$$

as required. For the universal introduction rule, i.e.

$$\frac{\Gamma, x : T \vdash \phi}{\Gamma \vdash \forall x : T \cdot \phi}$$

We may assume c exactly covers the conclusion. By induction, the premiss follows. Hence, by the quantifier rule in **CST_T**:

$$\frac{\Gamma^c, x : T \vdash \phi^{c,x:T}}{\Gamma^c \vdash (\forall x : T \cdot \phi)^c}$$

The introduction rule uses the substitution lemma for the translation. ■

Putting this together with part 4 of the previous proposition, we have:

THEOREM 6.5

CST is a conservative extension of **CST_T** relative to well-typed wff.

CST_T is conceptually prior in that it is a *typed logic* in the traditional and natural sense. It thus provides conceptual underpinning for the half-way house instantiated in **CST**. Consequently, we are justified in using our original cavalier and practical syntax together with the post-hoc type inference system. It all comes out in the wash. This is advantageous since we really have enough to do constructing the proofs without having to worry about type-checking.

7 Further work

CST is a core theory that provides a basic framework for exploring some fundamental issues about specification. Within it, a great many of the central questions about the foundations of specification can be addressed. Conservativity, polymorphism and type inference can all be articulated and studied. However, there is much left to do. Recursive specifications, and their role in specification, has not been touched upon. The impact of *computability* considerations has not been considered. Should we insist that specifications be computable? Does this restrict the expressive power of specification too much? Where does refinement come into the picture? We have not provided any set theoretic models of **CST**. The standard models are constructed in a familiar way, from the standard model of the natural numbers, by closure under set theoretic Cartesian products and finite subset construction. In addition there are a whole range of non-standard ones including those obtained from models of Peano arithmetic. Indeed, **CST** is a conservative extension of Peano arithmetic. Moreover, if the induction principles are restricted to Σ wff, it is a conservative extension of primitive recursive arithmetic: **CST** is a theory of data items, not a theory of infinite sets given in extension. Finally, we have yet to study the need and impact of new type constructors such as recursive types, sub-types, function space types, universes, objects/classes etc. Each of these requires a short paper of its own.

Acknowledgements

I would like to thank Martin Henson, Norbert Völker and Amnon Eden for detailed comments and discussion on this paper. Two anonymous referees also supplied very useful comments and advice.

References

- [1] J. R. Abrial. *The B-Book*. Cambridge University Press, Cambridge 1996.
- [2] H. P. Barendregt. Lambda calculus with types. In *Handbook of Logic in Computer Science*, S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum, eds., pp. 118–310. Oxford University Press, 1992.
- [3] J. Barwise. *Admissible Sets and Structures*. Springer Verlag, Berlin. 1975.
- [4] A. Blikle. Three-valued predicates for software specification and validation. In *VDM '88 VDM – The Way Ahead*, pp. 243–266. Springer-Verlag, 1988.
- [5] S. M. Brien and J. E. Nicholls. *Z Based Standard Version 1.0*, Oxford University Computing Laboratory. PRG 107, 1992.
- [6] J. H. Cheng and C. P. Jones. On the usability of logics which handle partial functions. In *Proceedings of the Third Refinement Workshop*, Springer-Verlag, 1990.
- [7] J. Dawes. *The VDM-SL Reference Guide*. Pitman. 1991.
- [8] A. Diller. *Z: An Introduction to Formal Methods*. John Wiley & Sons, 1990.
- [9] P. F. Gibbins. VDM: Axiomatising its propositional logic. *BCS Computer Journal*, **31**, 510–516, 1988.
- [10] M. Henson and S. Reeves. Revising Z -I semantics and logic. *Journal of Formal Aspects of Computing*, **11**, 359–380, 1999.
- [11] M. Henson and S. Reeves. Revising Z -II Logical Development. *Journal of Formal Aspects of Computing*, **11**, 381–401, 1999.
- [12] W. Hodges. *The Semantics of Specification Languages*. Queen Mary College, London. 1990
- [13] C. B. Jones and C. A. Middleburg. A typed Logic of partial functions reconstructed classically. *Acta Informatica*, **31**, 399–430, 1994.
- [14] C. B. Jones. *Systematic Software Development Using VDM*. Prentice Hall, Hemel Hemstead, 1986.
- [15] M. Kooij. Interface specification with temporal logic. *Proc. 5th International Workshop on Software Specification and Design*, pp. 104–110, May 1989.
- [16] P. G. Larson and W. Pawlowski. The formal semantics of ISO VDM-SL. *The Journal of Computer Standards and Interfaces*, 1995.
- [17] T. S. E. Maibaum and W. M. Turski. *The Specification of Computer Programs*. Addison Wesley. 1987.

- [18] R. Milne. The proof theory of the Raise specification language. Technical report. Raise/STC/REM/12/V3. England. 1990.
- [19] K. Middelburg. The logical semantics of flat VVSL. Technical Report, Neher Laboratories, Number 954 RNL/89, December 1989.
- [20] R. Moore. and B. Ritchie. *Proof in VDM-A practitioners Guide*, FACIT, Springer Verlag, ISBN 3-540-19813-X, 1994.
- [21] J. E. Nicholls, ed. *Z-Notation version 1.2*, 1995.
- [22] S. Owre, J. Rushby, S. Natarajan and F. Von Henke. Formal verification of fault tolerant architecture: Prolegomena to the design of PVS, *IEEE Transactions of Software Engineering*, **21**, 107–125, 1995.
- [23] S. Owre and S. Natarajan. The formal semantics of PVS. NASA/CR-1999-209321, May 1999.
- [24] B. Potter, S. Sinclair and D. Till. *An Introduction to Formal Specification and Z*. Prentice Hall, 1991.
- [25] V. Y. Sazonov. On bounded set theory, Invited Talk on the 10 international, Congress on Logic, Methodology and Philosophy of Science, Florence, August 1995. In *Volume 1: Logic and Scientific Method*, pp. 85–103. Kluwer Academic Publishers, 1997.
- [26] J. M. Spivey. *Understanding Z*. Cambridge University Press, 1988.
- [27] J. M. Spivey. *The Z Notation: A Reference Manual*. Prentice Hall, 1992.
- [28] G. Struth. A calculus for set-based program development. Part 1: mathematical foundations. Report 2003-15. 2003. Institut Fur Infomatik. D-86135 Augsburg.
- [29] A. Tarlecki, B. Konikowska and A. Blikle. A three-valued logic for software specification and validation. In *VDM '88 VDM – The Way Ahead*, pp. 218–242. Springer-Verlag, September 1988.
- [30] P. A. S. Veloso and S. R. M. Veloso. On methods for safe introduction of operations. *Information Processing Letters*, **64**, 231–128, 1997.
- [31] J. Woodcock and S. M. Brien. W: a logic for Z. In *Z Users Workshop*, York, 1991, *Proceedings of Sixth Annual Z User Meeting*, J. E. Nicholls, ed. Springer Verlag, 1992.
- [32] J. Woodcock and J. Davies. *Using Z- Specifications, Refinement and Proof*. Prentice Hall, 1996.

Received 16 February 2004