J. Symbolic Computation (1999) **11**, 1–29
Article No. jsco.1999.0347
Available online at http://www.idealibrary.com on IDEAL®

A Calculus for Schemas in Z

S. M. BRIEN† AND A. P. MARTIN‡

†*Mercer Management Consulting, Ltd., London, U.K.*
‡*Oxford University Software Engineering Centre, Oxford, U.K.*

pl.supply date

The popularity and flexibility of the Z notation can largely be attributed to its notion of schemas. We describe these schemas and illustrate their various common uses in Z. We also present a collection of logical laws for manipulating these schemas. These laws are capable of supporting reasoning about the Z schema calculus in its full generality. This is demonstrated by presenting some theorems about the removability of schemas from Z specifications, together with outline proofs. We survey briefly models against which this logical system may be proven sound, and other related logics for Z.

1. Introduction

The Z notation (Spivey, 1992b) has become a popular tool, both in teaching software specification and in industrial practice. Testimony to this may be seen in the regular Z User Meetings (Bowen *et al.*, 1997), and numerous tools and textbooks. Examples of the latter include those of Woodcock and Davies (1996) and Wordsworth (1993); there are at least 35 in total. Tools and publications are listed by Bowen (1996). This proliferation of interest has led to the development of an ISO/IEC standard for Z (Nicholls, 1995), the standardization panel having representatives from approximately ten countries. That Draft Standard will be our normative reference in this paper; usually it will be referred to simply as "the Standard", or "Standard Z".

A brief introduction to the Z notation is presented in Section 2. Z is based on classical first-order logic and set theory, but an integral part of the notation is its language of schemas. Much of the appeal of Z can be attributed to this schema notation, which provides a flexible and powerful means of structuring specifications. Z schemas are used in virtually every Z specification. These schemas are also described in Section 2, together with a discussion of their representation and syntax. A separate section (Section 4) describes some of the various applications of the schema notation in Z.

The language of schemas in Z includes a collection of operators on schemas—many related to similar logical operators. These have become known as the *schema calculus*. This is something of a misnomer, since in most use there has been no practical calculus associated with schemas, or at best, a semi-formal notion of expansion to a normal form and some *ad hoc* rules of manipulation.

Being a notation based on classical logic, Z provides many opportunities for proof of properties of specifications—either related to their internal consistency, or their conformance to some other specification, or their refinement towards an implementation in an imperative programming language. Perhaps surprisingly, there have been very few attempts to provide a comprehensive logic in which to prove such properties. Rules for

0747–7171/99/000001 + 29 $30.00/0

manipulating terms of logic and set theory are well known, but logics for the schema operators are not.

The contribution of this paper is to provide a true calculus for schemas, a component of a whole logic for Z. We present $\mathcal{V}$, a logic for Z which incorporates the ability to reason about schemas. We prove a result which demonstrates in the fullest generality that this logic is sufficient for reasoning about all uses of Z. This is accomplished by showing that the schemas may be eliminated from a Z specification. The possibility of removing schemas has been suspected—and even assumed—for some time; this result makes it precise. The soundness of $\mathcal{V}$ is proved elsewhere (Brien, 1998), with respect to a model theory consistent with that used in the Draft Z Standard.

The relation of *Z schemas* to schemas in other branches of computing may be rather tangential. One interpretation of certain Z schemas is as a specification of a procedure (Section 4.1); the schema might be regarded as a schematic form of the implementation. The schema is intended to capture at a high level *what* an implementation is to achieve, and not necessarily *how* that is to be done. Whilst this was one of the earliest interpretations of Z schemas, it is by no means the only one, as we shall explain.

1.1. OUTLINE OF THE PAPER

In this paper we present logical rules which can be used to manipulate instances of schemas in Z specifications. Whilst these rules are quite straightforward, ease of use comes at the expense of additional logical apparatus to manage names. Z schemas use names (of variables, etc.) in a way quite unusual in logic. The "Laws" presented below may be basic or derived—they are included either through relevance to the exposition or because they are used in the proofs of theorems. An appendix gives the entire set of basic rules.

After introducing Z, in Section 3 we begin by explaining the key features of the logic $\mathcal{V}$ for reasoning in Z. The full account of the logic is elsewhere; here we concentrate on the schema constructs. In consequence, whilst we include many properties of schemas, we have avoided labelling some as definitions and some as derived properties; instead, all are labelled as laws.

In Section 4 we survey most of the uses of schemas in Z, and how these schemas may be constructed and decomposed. A number of logical rules are given. The following section outlines some of the theorems about Z which follow from these rules. The key result is that schemas may be eliminated altogether from any given Z specification.

Section 6 surveys related work, including models in which this logic may be proven sound. The paper concludes with a summary and pointers to further work.

2. The Z Notation

The Z notation began in the work of Abrial and others in Oxford, and has evolved over the last 20 years (Brien, 1994) to cover a wide international community. To a large degree, it is simply a style for using first-order predicates and a typed form of Zermelo set theory. It uses a precise grammar to facilitate machine analysis—some of the most useful Z tools are type-checkers (for example, Spivey's 1992a, *f*UZZ). Many of the fundamental features of Z are also present in B (Abrial, 1996), with the notable exception of Z schemas.

Many organizations are using Z as a means of writing precise high-level descriptions of software systems. Most typically, a Z specification describes the *state* and *operations*

of a system (see below), and so it is well suited, say, to the description of secure data processing—but many other applications exist. Z includes a notion of *refinement* so that a more concrete specification can be proved consistent with a more abstract one. That feature will not be explored here.

2.1. Z mathematical language

The three main syntactic classes of Z are declarations, predicates, and expressions. Whenever a variable is declared, its type must be indicated. Fundamental types are either "given sets"—introduced into the specification without further analysis (an example is the set of integers, $\mathbb{Z}$)—or sets constructed from these using powerset and cross-product operators (or schema types; see below). Any set-valued expression may be used as a variable type. Type-checking rules of Z ensure that such sets are subsets of the fundamental types.

Reasonable declarations, then, include

$$x : \{1, 2, 3\}$$
$$x, y : \mathbb{N}$$

$\mathbb{N}$ is not a fundamental type, but is defined as

$$\mathbb{N} == \{x : \mathbb{Z} \mid x \geq 0\}$$

Here we see "==" used for definitional equality, and we see Z notation for a simple-set comprehension. Observe that this consists of a declaration part and a predicate part, separated by "|".

Predicates are constructed using the usual logical operators, quantifiers having the general form

$$\forall \mathsf{Declaration} \mid \mathsf{Constraint} \bullet \mathsf{Predicate}$$

(similarly for the existential quantifier). When the constraint is vacuous, it may be omitted. Thus, the following predicates are equivalent:

$$\forall x : \mathbb{Z} \mid x \geq 0 \bullet P$$
$$\forall x : \mathbb{N} \bullet P$$

When an identifier is declared at the outermost scope (an "axiomatic definition"; as distinct from a declaration within a quantifier, or a set comprehension), it is given some visual furniture to identify it: a vertical and horizontal line, with the declaration above the line, and a predicate constraining the value below.

$$\begin{array}{|l} succ : \mathbb{N} \to \mathbb{N} \\ \hline \forall n : \mathbb{N} \bullet succ(n) = n + 1 \end{array}$$

The above definition comes from the *Z Mathematical Toolkit*, a large collection of definitions covering the theory of sets, relations, functions, numbers, sequences, etc., including for example, the definition of $\mathbb{N}$, above. The toolkit allows Z users to communicate reasonably complex ideas using shared common definitions. It is documented in Spivey (1992b) and Nicholls (1995).

2.2. Z schema language

The language introduced thus far does not afford much scope for structuring specifications. Z language of schemas allows declarations and predicates to be associated together to form a new syntactic entity. The general form of a schema, then, is

$$[\mathsf{Declaration} \mid \mathsf{Predicate}]$$

The predicate part may be omitted.

The next Z paragraph contains a declaration of a schema named *File*, which has two components. One is a sequence of integers which represents its contents; the other is a natural number recording the file size. The second component is redundant in that its value can be discovered from the first. It is often useful to include redundant information in a specification (for ease of reading, and because checking that redundant information remains consistent often provides a useful cross check that the specification behaves is as expected). Here the predicate part of the schema asserts a state invariant for the file; the guarantee that the *size* component always correctly records the size of the *contents*.

$$File == [contents : \operatorname{seq} \mathbb{Z};\ size : \mathbb{N} \mid size = \#contents].$$

Z uses schemas to denote structured types, system states, operations on those states, and in many other roles. These are outlined in Section 4. Thus the question of what a schema "means" depends very much on the context of its declaration/use. A denotational semantics for schemas however, is well accepted: a schema denotes a set of *bindings* of appropriate type, having components which satisfy the relevant predicate. The relationship between schemas and bindings is explored in Sections 4.3 and 4.4. This means of giving semantics to schemas is well-documented in the Z Standard; the logical consequences of so doing are reported in this paper.

Schemas are more commonly written using a display notation similar to that for axiomatic definitions given above. The form most familiar to users of Z is shown below. This is visually distinctive, and helps to identify a Z document. The following Z paragraph means exactly the same as the preceding one.

File

$$\begin{array}{|l} contents : \operatorname{seq} \mathbb{Z} \\ size : \mathbb{N} \\ \hline size = \#contents \end{array}$$

In both of these forms, we have the schema appearing as an *expression*, and being given a name. The Standard uses "==" for all forms of definitional equality; the symbol $\widehat{=}$ has previously been used in schema definitions (Spivey, 1992b). A Z specification consists of a sequence of Z paragraphs (with commentary), such as those shown here. There are other Z paragraph forms, but they will not be relevant to our present discussion.

It is important to note that whilst in early accounts, the schemas were largely viewed as "macros" (they enabled a name to be given to a collection of declarations and predicates), by the time Spivey (1988) wrote his seminal account of Z semantics, schemas were entities in their own right. Z incorporates a collection of operators for combining schemas in various ways (see below)—these operators have been known as the schema calculus. It is this treatment of schemas as first-class entities which makes reasoning about them a significant challenge—and this is the issue addressed by this paper.

3. The $\mathcal{V}$ Logic

The schema properties given in this paper are axioms and theorems of a logic developed for reasoning about the consequences of Z specifications, named $\mathcal{V}$. It is presented in the form of a typed sequent calculus with structured antecedents. We shall describe the structure of the $\mathcal{V}$ sequent and the operation of the rules of the logic.

The distinctive feature of the logic is that, in order to treat schemas faithfully, it is necessary to include a "scope calculus". In conventional logics, the property of a variable name being bound or free in a particular predicate or expression is easily, statically determined. In Z, freeness also depends on the *context* of the term—therefore Z notion of freeness is different from that classically used, and must be carefully defined by $\mathcal{V}$.

3.1. SEQUENTS

A sequent of $\mathcal{V}$ is an assertion about well-formed and well-typed fragments of Z. Z has a simple decidable type system (Spivey and Sufrin, 1990), and so in all that follows, well-typedness will be assumed. A traditional sequent is composed of an antecedent and a consequent, where the antecedent records the assumptions under which the consequent is entailed. For $\mathcal{V}$ this structure must be such that it can express all the assertions that we wish to make about specifications, and all the assertions we need during the intermediate stages of a proof.

In a traditional sequent calculus, both the antecedent and the consequent consist simply of lists of predicates. However, in $\mathcal{V}$ we also include declarations such as schemas in the antecedent whose syntax corresponds to a specification (i.e. a sequence of Z paragraphs). The entailment relation is between an antecedent specification and a single consequent predicate. Thus a well-formed *Sequent* has the following structure:

$$Spec \vdash Pred$$

In the logic, we use a horizontal (compact) style of presenting a specification, because it suits the sequent form. The compact and display forms of Z schemas are explained in Section 2.2. The term *Spec* will consist of Z paragraphs separated by "|". We use this symbol because it corresponds to its use in the vertical presentation of specifications to identify a new definition which introduces a fresh scope. The comma (more commonly used in sequent calculi) has significance as a list separator in Z.

In a usual sequent calculus the order of the clauses in the antecedent is not significant: they are all predicates and are interpreted in the same scope. In contrast, the clauses in a $\mathcal{V}$ antecedent are Z paragraphs, including declarations, which create fresh nested scopes stretching rightwards through the sequent. Thus the interpretation of an individual clause in a $\mathcal{V}$ sequent is dependent on its context. Hence, the order of the clauses of the antecedent in $\mathcal{V}$ is significant.

Using the model described in Section 6.1, we determine that a well-formed sequent is *valid* if the consequent predicate is true for every interpretation of the antecedent specification.

RULES

A sequent calculus consists of a number of rules for manipulating constructs within its sequents, as well as structural rules. $\mathcal{V}$ has rules for manipulating the constructs of

the predicate calculus in both the antecedent and in the consequent. As is conventional, we present rules using a horizontal bar, with a (possibly empty) list of sequents above the bar, and the conclusion below. Rules in $\mathcal{V}$ do not contain side-conditions because the constraints (such as those on free variables) are expressed in terms of premises. We extend the definition of a sequent to include the notion of a constraint that is evaluated in the context of a specification. These constraints are decidable in $\mathcal{V}$.

$$\mathit{Spec} \vdash \mathit{Constraint}$$

For example, $\Gamma \vdash x \backslash P$ asserts that in the context of the specification Γ, variable x does not occur free in P. In many contexts, the calculation of freeness will be just as in conventional logics. Where a schema is involved, however, different conditions will apply.

The rules of $\mathcal{V}$ for the propositional operators in Z, and for the ordinary expressions (powerset, set comprehension, tuple, etc.) are entirely as one would expect. In this paper we limit our attention to rules relating to Z schemas. These are incorporated in the sequel as "Laws"—whether primitive or derived rules. The full set of primitive rules is presented in an appendix, and the reader will observe that the rules for the classical logical and set theoretic constructions are exactly as one would expect in a sequent calculus. A fuller account may be found in Brien (1998); a similar logical system appears in the Z Standard (Nicholls, 1995).

In the following presentation, where context is unimportant, we give simply equations or logical equivalences; otherwise full sequents are given. Where possible these properties are presented as axioms, but some must be inference rules, due to our treatment of "not free in" constraints.

4. Roles for Schemas in Z

In this section, we consider a number of different ways that schemas are used in Z, and laws which relate them.

4.1. STATE-BASED SYSTEMS

The most common use of schemas in Z is in the specification of state-based systems. Having defined a file as above, we might proceed by declaring operations which append input data ($\mathit{extra}?$) to the file and report the contents of the file as an output ($\mathit{data}!$).

Append

File
File'
$\mathit{extra}? : \operatorname{seq} \mathbb{Z}$

$\mathit{contents}' = \mathit{contents} \frown \mathit{extra}?$

Read

File
File'
$\mathit{data}! : \operatorname{seq} \mathbb{Z}$

$\mathit{contents}' = \mathit{contents} = \mathit{data}!$

Append is a schema which denotes not the state of some system, but an operation upon that system. Two copies of the *File* schema are brought into scope; the second one has all its components primed. The unprimed components belong to the state before the operation has taken place; the primed components describe the state after the operation. Inclusion of *File* brings both its declarations and predicates; thus not only are *contents* and *size* brought into scope, the state invariant which relates them is part of *Append*

also. Likewise, an invariant relating $contents'$ and $size'$ is also present in *Append*, and so there is no need to state explicitly how $size'$ relates to *size*.

This style of specification is so common that a shorthand is used to cover the inclusion of both *File* and *File*′. The following schemas are assumed declared:

$$\Delta File == [File;\ File']$$
$$\Xi File == [File;\ File' \mid \theta File = \theta File']$$

The first of these is used in operations like *Append* where the state changes; the second incorporates a predicate which asserts that no change takes place—it is used in the definition of functions which query the state without altering it, as in *Read*. The θ will be explained in Section 4.6. In this case, $\theta File = \theta File'$ is equivalent to $contents = contents' \land size = size'$. Thus the schemas above would more usually appear as:

Append

$\Delta File$

$extra? : \operatorname{seq} \mathbb{Z}$

$contents' = contents \frown extra?$

Read

$\Xi File$

$data! : \operatorname{seq} \mathbb{Z}$

$data! = contents$

We mention this and other alternative presentations to emphasize that a comprehensive logic for schemas in Z must cope with a broad range of notations and conventions, including multiply nested schema declarations.

In a state-based system it is also important to define how the system is to be initialized. This may be described using a degenerate operation, which has an "after" state, but no "before" state:

InitFile

$File'$

$contents' = \langle\rangle$

Specifications in this style can be refined towards code in a very natural way (the refinement is typically informal, but can be made formal (Calvacanti, 1997)). The usual expectation is that the state variables will become global variables in some program (or module, or class). The initialization schema will be refined either by initialization of the variables by the compiler, or by some explicit initialization procedure (or method). The operation schemas will be refined by procedures (or functions, if they do not change the state, or methods) with appropriate inputs and outputs.

POINTS TO NOTE

Even in this elementary Z specification, we see a number of different uses of schemas, and an expectation that they will denote different kinds of artifacts in the implementation of the system being specified.

Each of the boxes above defines a named schema. Some of the schema names have been duplicated for pedagogical reasons; the result is that this document is not a valid Z specification. Besides forming expressions, we have also seen schemas used as declarations. This idea will be explored further, below.

We also implicitly have two different levels of semantic descriptions. The schemas, we have hinted, are first-class objects, and so have a uniform semantic description (outlined

below). They also contribute in different ways to a system specification (as descriptions of state, of initialization and of particular operations), and so, at another level, belong to quite a different model. We consider both models in Section 6.1.

4.2. SCHEMAS AS DECLARATIONS

Z style for quantifiers, set comprehensions, etc. borrows from the schema style. If we were to construct a sub-type of the type of *File* where the *contents* component is to be sorted in nondecreasing order, we might write

SortedFile

$$File$$

$$\forall i, j : \operatorname{dom} contents \mid i < j \bullet contents(i) \leq contents(j)$$

The reader will recognize a schema between the $\forall$ and the $\bullet$ symbols. We could have defined that schema explicitly, and then used it to write the predicate in a short form. Within the schema above we could have written:

$$\textbf{let } S == [i, j : \operatorname{dom} contents \mid i < j] \bullet \\ \forall S \bullet s(i) \leq s(j)$$

A calculus of schemas must allow for all such alternative forms of expression, and permit proofs of equivalence where appropriate.

POINTS TO NOTE

If *contents* were for some reason defined globally instead of (or as well as) as a state component, we could have defined S at the top level

$$S == [i, j : \operatorname{dom} contents \mid i < j]$$

In this case, any use of S would refer to that global instance of *contents*, so writing $(\forall S \bullet \ldots)$ in *SortedFile* above would not have had the desired effect. In short, schemas may not be considered as macros.

The use of a schema declaration constrained by a predicate can be eliminated using the following law.

LAW 4.1. $(\forall S \mid P \bullet Q) \Leftrightarrow (\forall S \bullet P \Rightarrow Q)$

4.3. SCHEMAS AS RECORD TYPES

An alternative (or additional) use for the schema *File* we have declared above is as the definition of a collection of bindings, or a *record type*, as in many high-level programming languages. We may see *File* as declaring a type for variables so that a declaration

$$f : File$$

would declare f as a variable having two components: $f.contents$, a sequence of integers, and $f.size$, a natural number. *File* is more constrained than a usual record type in that it also contains a predicate which will ensure in this case that the identity $\#f.contents = f.size$ holds, no matter what value f may take.

We could use *File* in this way to declare a directory of files:

$$
\begin{array}{|l}
\textit{Directory} \\
\hline
\textit{lookup} : \textit{Name} \pfun \textit{File} \\
\textit{info} : \textit{Name} \pfun \mathbb{N} \times \textit{Date} \\
\hline
\operatorname{dom} \textit{lookup} = \operatorname{dom} \textit{info} \\
\forall\, n : \operatorname{dom} \textit{lookup} \bullet (\textit{lookup}\ n).\textit{size} = (\textit{info}\ n).1 \\
\hline
\end{array}
$$

This definition provides two partial functions *lookup* and *info*, which, given a file name, return respectively either the contents of the file, or some directory information for the file. The state invariants ensure that both functions have the same domain (that is, that information is stored for all the known files, and no others), and that the size information stored in the *info* lookup is an accurate record of the size of the corresponding file.

Instead of the function *first* which Spivey (1992b) uses to select the first component of a tuple, the Standard uses a postfix ".1". The range types of *lookup* and *info* illustrate the relationship between schema types and Cartesian products. Both look similar: in the case of a Cartesian product, the *position* is important; in the case of a schema type (labelled product), the *label* or component name is important *and the position in the declaration list is not.*

POINTS TO NOTE

Here we have the kernel of the idea for the semantics of schemas. A schema expression denotes the set of bindings which have the schema's signature and satisfy the property of the schema. The names of the components in the signature are important; they form part of the schema's type. It follows that the next schema is quite different from *File*. Indeed, the two are incomparable since their types differ.

$$
\begin{array}{|l}
\textit{File2} \\
\hline
\textit{data} : \operatorname{seq} \mathbb{Z} \\
\textit{size} : \mathbb{N} \\
\hline
\textit{size} = \#\textit{data} \\
\hline
\end{array}
$$

Viewing a schema as a collection of bindings, we may state some identities relating schemas presented in the style we have seen (declaration constrained by predicate):

LAW 4.2. $[(S \mid P) \mid Q] = [S \mid P \land Q]$

LAW 4.3. $[S \mid P] \subseteq S$

LAW 4.4. $S = T \Rightarrow [S \mid P] = [T \mid P]$

For example, we have $\textit{OrderedFile} \subseteq \textit{File}$, and whereas from Law 4.4, *OrderedFile* is equivalent to

$$[(\textit{contents} : \operatorname{seq} \mathbb{Z};\ \textit{size} : \mathbb{N} \mid \textit{contents} = \#\textit{size}) \mid (\forall\, i, j : \mathbb{Z} \mid \cdots \bullet \cdots)]$$

this is, by Law 4.2:

$$[\textit{contents} : \operatorname{seq} \mathbb{Z};\ \textit{size} : \mathbb{N} \mid \textit{contents} = \#\textit{size} \land (\forall\, i, j : \mathbb{Z} \mid \cdots \bullet \cdots)]$$

Such equivalences are understood by Z users; this calculus makes them precise.

4.4. BINDINGS AND SUBSTITUTIONS

The account of Spivey (1992b) does not allow an explicit representation of a binding in extension within Z. It is convenient to do so, however, so the Draft Standard provides a syntax for this. Using this notation, one member of *File* might be

$$\langle\!| contents == \langle 4, -2, 36\rangle, size == 3 |\!\rangle$$

An unexpected feature of the particular treatment of names and bound/free variables of Z (see below) is that substitution can be expressed *within the language* using a binding. The Standard uses the symbol $_\odot$ to denote the use of a binding as a substitution into a predicate (and $_\odot$ for substitution into an expression) though this can be defined away in a similar way to **let** (Spivey, 1992b, pp. 59, 71). The choice of symbol is deliberately reminiscent of the dot used to indicate selection:

$$\langle\!| contents == \langle 4, -2, 36\rangle, size == 3 |\!\rangle.size$$

takes the value 3; the predicate

$$\langle\!| contents == \langle 4, -2, 36\rangle, size == 3 |\!\rangle_\odot (size < max)$$

may be simplified to $3 < max$. Tables for computing these substitutions are presented in the Standard; they may be derived from the rules in the Appendix.

POINTS TO NOTE

Besides this notion of substitution, Z has long had a notation for schema component renaming. $S[x/y]$ is the schema S with all references to component x replaced by component y. There are two types of variable in Z: free variables and labels (schema/binding components). Labels can bind other variables. This is the key distinction between Z and other logical systems. We have made variables (as labels) first-class objects. The relationship between free variables and labels is largely orthogonal as characterized by the following law:

LAW 4.5. $b \odot (S[x/y]) = (b \odot S)[x/y]$

The theory of renaming is outside the scope of this paper, but is included in a full account of $\mathcal{V}$ (Brien, 1998). It is developed in a similar way to the theory of substitution.

The following identity shows how to convert the use of a schema as a declaration into a schema used as an expression, by changing a quantification over the components of the schema into a quantification of a binding.

LAW 4.6. $$\frac{\Gamma \vdash b \backslash (\forall S \bullet P)}{\Gamma \vdash \forall S \bullet P \Leftrightarrow \forall b : S \bullet b_\odot P}$$

Using bindings, we may make precise the semantics of the usual schema presentation of a declaration constrained by a predicate. This law and the preceding one can be used to prove Law 4.1.

LAW 4.7. $b \in [S \mid P] \Leftrightarrow b \in S \land b_\odot P$

The following equivalence is perhaps one of the most surprising consequences of this treatment of substitution:

LAW 4.8. $$\frac{\Gamma \vdash S \backslash S}{\Gamma \vdash b \in S \Leftrightarrow b_\odot S}$$

This property relates schemas used as expressions to schemas used as predicates. The constraint $S \backslash S$ may appear odd: it says that the component variables of S do not occur free in S (i.e. do not occur in the constraining sets in the declaration part of S).

In some of these rules we have seen freeness constraints which are not as straightforward as those encountered previously. By $S \backslash T$ we mean (for S and T schemas) that none of the components of S occur free in T. See Section 4.8.

4.5. SCHEMA CALCULUS

A more practical (easier to implement) specification of *File* might have placed an upper limit on the size of *contents*. Consider this definition:

File

$$\begin{array}{|l} contents : \operatorname{seq} \mathbb{Z} \\ size : \mathbb{N} \\ \hline size = \# contents \\ size < 10000 \end{array}$$

In the state-based system Z specification style, under this definition, the *Append* operation of Section 4.1 would be partial. It could be guaranteed to complete only when the combined size of the *contents* and the material to be appended did not exceed the chosen maximum. We may compute the precondition of the operation by quantifying over the "post-state" variables.

$$\mathbf{pre}\ Append = \exists\, contents' : \operatorname{seq} \mathbb{Z}, size' : \mathbb{N} \bullet Append$$

pre *Append* is a schema. It may be rewritten using the laws given in this paper as follows:

$$[File;\ extra? : \operatorname{seq} \mathbb{Z} \mid size + \# extra? < 10000]$$

Because such partial operations are very common in state-based specifications, and because it is usually prudent to specify failing behaviours as well as successes, it is customary to augment specifications with material for reporting success or failure.

$$Report ::= Ok \mid TooBig$$
$$Success == [r! : Report \mid r! = Ok]$$
$$FailTooBig == [\Xi File;\ r! : Report \mid size + \# extra? \geq 10000 \land r! = TooBig]$$
$$RobustAppend == (Append \land Success) \lor FailTooBig$$

The precondition of *FailTooBig* is the complement (modulo the state invariant) of that of *Append*. Moreover, it uses $\Xi File$ to indicate that the file state is unchanged in this failure case. The final schema *RobustAppend* performs the append operation if possible, reporting failure otherwise.

POINTS TO NOTE

In this section we have seen operators of the *schema calculus*. Operators (quantification, disjunction, conjunction) which typically belong to the logical calculi are here used upon schemas (as expressions), the result being other schemas (as expressions).

When schemas constructed using schema propositional operators are used as predicates, the schema calculus operators may be replaced by logical ones. Z overloads the operator symbols, so in the laws below, the operator on the left is a schema calculus operator; that on the right is a logical operator, as is the equivalence symbol.

LAW 4.9. $[S \land T] \Leftrightarrow S \land T$

LAW 4.10. $[\neg\, S] \Leftrightarrow \neg\, S$

Square brackets are used in these laws to disambiguate the overloaded symbols: since square brackets surround schemas (and $[S] = S$) the term inside the brackets must be a schema, so the instance of $\land$ or $\neg$ there must be a schema calculus operator, and not a logical one.

Propositional operators in schemas used as expressions may also be removed, using the following laws:

LAW 4.11. $$\frac{\Gamma \vdash S \land T \backslash S \land T}{\Gamma \vdash b \in S \land T \Leftrightarrow b_{\odot} S \land b_{\odot} T}$$

LAW 4.12. $b \in \neg\, S \Leftrightarrow b \notin S$

We may also say what it means for a binding to belong to a schema quantification (where the quantified schema is used as an expression): the schema quantification is replaced by a logical quantification, with the binding used as a substitution.

LAW 4.13. $$\frac{\Gamma \vdash b \backslash (\forall\, S \bullet T) \qquad \Gamma \vdash S \backslash T}{\Gamma \vdash b \in (\forall\, S \bullet T) \Leftrightarrow (\forall\, S \bullet b_{\odot} T)}$$

Unlike the definition of **pre** *Append*, the *initialization theorem* is not an instance of a schema quantification. This theorem is a result often considered for state-based Z specifications; proving it demonstrates something about the consistency of the specification.

$$\exists\, File' \bullet InitFile$$

This is a logical quantification. The result is expected to be *true*, not some schema. Thus, in this case, *InitFile* is playing the role of a predicate, the predicate

$$contents' \in \operatorname{seq} \mathbb{Z} \land size' \in \mathbb{N} \land size' = \# contents' \land contents' = \langle\, \rangle$$

4.6. THETA

The basis of the meaning of a schema predicate is through that of a theta-term. In this section we define the meaning of a theta term and show how it relates to an identity substitution.

The theta term θS is a binding whose component names are those of the schema S. It identifies its components with the values of the same names in the local environment. Typically, θS will occur only in schemas where S is part of the declaration (such as $\theta File$ in $\Delta File$ above), but nothing forces this to be the case. This binding is the identity binding, mapping each name to its value. Hence, we define its characteristic property as follows:

LAW 4.14. $\theta S.x = x$

This holds only if x is a component of S, but that is guaranteed by the type system.

The binding defined by a theta term is determined by the type rather than the value of the defining schema. Therefore, we can derive the following result ($S \subseteq T$ is one way to assert that S and T both have the same type):

LAW 4.15. $S \subseteq T \vdash \theta S = \theta T$

Since a theta term identifies its component names with the same variable names in the local context, it behaves as an identity substitution:

LAW 4.16. $(\theta S) \odot e = e$

LAW 4.17. $(\theta S) \odot P \Leftrightarrow P$

Since a theta term is an identity substitution, a substitution into it either absorbs the theta term where b and S have the same components (guaranteed by the typing rules):

LAW 4.18. $b \odot \theta S = b$

or it distributes through it where b and S have disjoint components as in Law 4.21 below. When the component names overlap, then the substitution $b \odot \theta S$ corresponds to the substitution b restricted to the components of the schema S. The following derived rule shows how a substitution can be restricted to those variable names that are free in the predicate.

LAW 4.19.
$$\frac{\Gamma \vdash b \setminus \exists S \bullet P}{\Gamma \vdash b \odot P \Leftrightarrow (b \odot \theta S) \odot P}$$

We can eliminate all instances of theta terms by replacing them by an explicit binding.

LAW 4.20. $\theta S = \langle\!| x_1 == x_1, \ldots, x_n == x_n |\!\rangle$

This equality helps to illustrate that the free variables of a theta term include the component names of the defining schema.

A substitution distributes through a theta term, providing the substituted variable is not a component name of the schema:

LAW 4.21. $$\frac{\Gamma \mid x == e \vdash S \backslash x}{\Gamma \vdash \{\!|x == e|\!\}_{\odot} \theta S = \theta(\{\!|x == e|\!\}_{\odot} S)}$$

A schema is true as a predicate whenever the local variables corresponding to its component names satisfy the property of the schema. For a simple schema this can be stated as

LAW 4.22. $[x : s] \Leftrightarrow x \in s$

More generally,

LAW 4.23. $[S] \Leftrightarrow \theta S \in S$

This definition is the crucial link between the interpretation of a schema as a predicate and a schema as a set of bindings.

4.7. PROMOTION

We may use the material of the preceding sections together to introduce a higher level of structuring into our specification. We have seen how operations can be defined which update and/or query the state of *File*. We have also seen how to use *File* in the description of a more complex artifact *Directory*. We would like to be able to use the operations defined on *File* to update and/or query items within the structure *Directory*.

One popular way to do this uses *promotion*. We begin by using a schema to describe the use of a change to a certain named file as a change to the whole directory.

Promote

$\Delta Directory$
$\Delta File$
$filename? : Name$

$filename? \in \mathrm{dom}\, lookup$
$\theta File = lookup\ filename?$
$\theta File' = lookup'\ filename?$
$\{filename?\} \lhd lookup = \{filename?\} \lhd lookup'$
$dir = dir'$

We can now define an append operation on an arbitrary named file

$$AppendToFile == \exists \Delta File \bullet Append \land Promote$$

and likewise the read operation.

$$ReadFile == \exists \Delta File \bullet Read \land Promote$$

This approach extends the "state and operations" style for using Z. It is almost object-oriented in its approach (some would say object-based). There are a number of object-

oriented extensions to Z (Stepney *et al.*, 1992) which develop these themes further. Promotion is important because it stays within Standard Z but makes considerable use of the schema apparatus to perform reasonably elaborate structuring.

Note that it is not possible to define within Z a function which would take an arbitrary *File* operation and return a promoted version. This is because Z is not a higher-order logic; the type system prevents the construction of such arbitrary functions. In Standard Z, however, one could define a promotion function for all operations of a uniform signature. Suppose there were a class of operations like *Read*. We could define

$$ReadClass == [\Xi File;\ data! : \operatorname{seq}\mathbb{Z}]$$
$$PromotedReadClass == [\Xi Directory;\ filename? : Name;\ data! : \operatorname{seq}\mathbb{Z}]$$

$$promoteReadClass : ReadClass \rightarrow PromotedReadClass$$
$$\forall S : \mathbb{P}\, ReadClass \bullet promoteReadClass\ S = \exists \Delta File \bullet S \land Promote$$

4.8. ON FREE VARIABLES

One consequence of the abstraction power of the schema calculus is that it is possible to have formulae containing schemas whose free variables are not visible from the text. That is, a predicate may constrain the value of a variable without it appearing in the text of the predicate.

For example, if *File* appeared as a predicate, *size* would be a free variable of that predicate. *File* as an expression, however, would have different properties. Moreover, since schemas may be defined at different levels of scope (e.g. using **let**, as above), the calculation of free variables is not a trivial one.

We have seen that $\mathcal{V}$ uses the constraint form $\Gamma \vdash x \backslash P$ to require that x not be free in predicate P in the context of Γ. A collection of rules is provided, for predicates and expressions, to permit the simplification of such constraints. These may be used to describe a complete decision procedure for those constraints. Some simple examples are:

$$\frac{}{\Gamma \vdash x \backslash y} \qquad \frac{\Gamma \vdash x \backslash e \qquad \Gamma \vdash x \backslash s}{\Gamma \vdash x \backslash e \in s}$$

To determine whether or not x occurs free in a quantified predicate, there are two cases to consider. Either x occurs nowhere at all, or it is one of the components of the schema in the quantification. We write $\Gamma \vdash x \prec S$ to denote the judgement that x is a component of schema S, in the context of specification Γ.

$$\frac{\Gamma \vdash x \backslash S \qquad \Gamma \mid S \vdash x \backslash P}{\Gamma \vdash x \backslash \exists S \bullet P} \qquad \frac{\Gamma \vdash x \backslash S \qquad \Gamma \vdash x \prec S}{\Gamma \vdash x \backslash \exists S \bullet P}$$

Observe that in the first case, schema S is appended to the specification Γ in the second subgoal.

The context is used in determining the truth of the $x \prec S$ statements. Such a judgement is converted into a statement about the schema *type* (that it is of type set ($\mathcal{P}$) of labelled product (Σ), with components x_i having type τ_i). Rules for determining types, within

the context of a specification, are given in the Z Standard.

$$\frac{\Gamma \vdash S \mathrel{⦾} \mathcal{P}\Sigma(x \rightsquigarrow \tau, x_1 \rightsquigarrow \tau_1, \ldots, x_n \rightsquigarrow \tau_n)}{\Gamma \vdash x \prec S}$$

Where the term on the left of the "\" is a schema, the judgement is replaced by a collection of judgements ensuring that none of the components of the schema is free in the relevant term (expression or predicate).

$$\frac{\Gamma \vdash x_1 \backslash P \qquad \ldots \qquad \Gamma \vdash x_n \backslash P \qquad \Gamma \vdash S \mathrel{⦾} \mathcal{P}\Sigma(x_1 \rightsquigarrow \tau_1, \ldots, x_n \rightsquigarrow \tau_n)}{\Gamma \vdash S \backslash P}$$

4.9. OTHER USES OF SCHEMAS

The foregoing account has concentrated on the main uses of schemas in Z. It must be stressed, however, that as simple associations of declarations and predicates, schemas find a wide range of uses in different specification styles. Stoddart (1997) uses schemas in process algebra-like specifications within Z, for example, augmenting the conventions for state-based systems described above. Fidge *et al.* (1998) use schemas to capture properties of components of dynamic systems, and the schema calculus to compose these, without use of the more familiar style for state-based systems.

4.10. SUMMARY

We have seen in this section that schemas are used in most parts of the Z notation in a variety of ways. These, however, all fall into the categories of declarations, predicates, or expressions. Laws have been presented which relate various schema constructs together, and permit, under certain side conditions, schema uses in one of these forms to be transformed into another. By careful use of renaming, these side conditions can be satisfied. A theory of renaming is included in (Brien, 1998), but we have not reproduced it here. As with the scope calculus for free variables, it must be constructed carefully to deal with the unusual semantics of Z schemas.

The semantics and logic discussed here and in the following section is entirely capable of dealing with these uses of schemas. We might be tempted to describe this as a *denotational* semantics for schemas, and distinguish it from an *operational* semantics for the state and operations style, but to use these terms might be misleading, since those two views are describing different aspects, and not different views of the same semantic features.

5. Results

We illustrate the effectiveness of the laws given previously by stating some meta-theoretical results, with sketch proofs.

The use of bindings as substitutions is useful in explaining the properties of schemas, but is not essential. That is, predicates of the form $b \odot P$ can be rewritten using an explicit substitution $\{\!| x_1 == e_1, \ldots, x_n == e_n |\!\} \odot P$, which might traditionally (outside Z) be written $P[x_1 \backslash e_1, \ldots, x_n \backslash e_n]$.

THEOREM 5.1. *Every instance of a binding used as a substitution in a type-satisfiable*

predicate can be replaced by a binding extension and hence can be replaced by a traditional substitution.

PROOF. If the formula in question is type satisfiable, then the type of the binding, and hence its set of component names, can be determined. The rule of Leibniz for substitutions

LAW 5.1. $$\frac{\Gamma \vdash e \odot P \qquad \Gamma \vdash b = e}{\Gamma \vdash b \odot P}$$

allows a binding used as substitution to be replaced by a binding extension. From that derivation, we can construct the following rule:

$$\frac{\Gamma \vdash \langle\!| x_1 == b.x_1, \ldots, x_n == b.x_n |\!\rangle \odot P \qquad \Gamma \vdash b = \langle\!| x_1 == b.x_1, \ldots, x_n == b.x_n |\!\rangle}{\Gamma \vdash b \odot P}$$

By using this rule all instances of bindings as substitutions (whether into expressions or into predicates) can be replaced by substitutions whose form corresponds to the classical, and hence removable, structure.□

The following theorem shows that quantifications of the form $\forall S \bullet P$ can be rewritten in the form $\forall x : e \bullet P$ etc.

THEOREM 5.2. *Every instance of a schema used as a declaration can be replaced by a declaration containing only schema expressions and bindings as substitutions.*

PROOF. (Outline) Logical rules not presented here (but included in the Appendix) allow us to transform each declaration in the antecedent of a sequent into a quantification. Therefore, if we can replace each schema declaration in a quantification, we can replace all instances of schema quantification. Each existential quantification can be transformed into a universal quantification by de Morgan's correspondence, suitably generalized for schemas:

$$\exists S \bullet P \Leftrightarrow \neg\, \forall S \bullet \neg\, P$$

Finally, by applying Definition 4.6, all universal quantification involving schemas can be replaced by set-bounded quantification and substitution.□

As we have seen, schemas S and T can be combined using schema conjunction to form a new schema $S \wedge T$. If this schema is used as a predicate, the schema operators can be replaced by logical ones.

THEOREM 5.3. *All instances of schemas used as predicates, constructed using schema propositional operators, can be eliminated.*

PROOF. By Laws 4.9, 4.10, etc., each of the schema propositional operators can be replaced by their logical equivalent.□

Theta provides a succinct notational device, but it is not essential.

THEOREM 5.4. *Every instance of a theta-term in a type-satisfiable formula (predicate or expression) can be eliminated.*

PROOF. Since we are working in a typed world, the type of each theta term can be determined. Therefore, each theta term can be replaced by its binding extension equivalent, using Law 4.20.□

Schemas used as expressions (as in $b \in S$) may be replaced by schemas used as predicates. This is not a trivial consequence of Law 4.8 because of the constraint on that law.

THEOREM 5.5. *All formulae containing instances of schemas used as expressions can be replaced by ones containing schemas used as predicates only.*

PROOF. (Outline) Any instance of a schema expression can be transformed using the rule of Leibniz and the extensionality of sets into a predicate of the form $b \in s$. If the free variables of the schema construction are disjoint from its component variables then we can apply Law 4.8 to transform the membership predicate into a substituted schema predicate. If there is a name clash, then it is necessary to rename the components of the schema and binding before transforming (details omitted).

Having eliminated all name clashes, the earlier transformation rule can be applied to replace the membership relation with a substituted schema predicate with the following form:

$$(b[x/y])_{\odot}[S[x/y]]$$

in which both the substitution and the schema are renamed.

We have placed no constraints on the relationship between the free variables and component names of the schema. The derivation is straightforward for the case where there is no clash and Law 4.8 can be applied directly.□

When schema calculus operators are used in an expression, they can be eliminated, as the next two theorems show.

THEOREM 5.6. *All instances of schema propositional operators can be eliminated from a formula.*

PROOF. (Outline) By Theorem 5.5, all formulae with schemas as expressions can be converted to ones using only schema predicates. Any renaming of these schemas can be distributed inwards through the schema propositional operators (the theory of renaming is not covered in this paper). Finally, by Theorem 5.3, the schema propositional operators can be replaced by their logical equivalents.□

THEOREM 5.7. *Every formula containing a schema quantification can be removed.*

PROOF. By Theorem 5.5, all formulae with schemas as expressions can be converted to ones using only schema predicates, likewise for declarations. Any renaming of these schemas can be distributed inwards through the schema quantification. Finally, the schema quantifiers can be replaced by their logical equivalents.□

The following theorem follows directly from Law 4.23.

THEOREM 5.8. *Every instance of a schema predicate can be replaced by a predicate containing schemas only as expressions.*

The theorems proved above may be combined to produce a significant result for this schema calculus. Reference to typed set theory in the statement of the theorem is of course a set theory which includes bindings/labelled products. With the removal of the schema apparatus, these labelled products are exactly analogous to Cartesian products, and only need an analogous collection of inference rules.

THEOREM 5.9. *The schema calculus we have defined is an assertion complete with respect to a typed set theory. Every formula containing a schema construct can be replaced by one without.*

PROOF. There are four steps to this process. They rely on the presence of typing information about the terms that are manipulated:

Removal of theta-terms. All instances of theta terms can be replaced by explicit identity bindings whose component names are the same as those of the defining schemas.

Conversion into schema predicates. All instances of schemas as declarations can be replaced by set-bounded declarations with schema expressions. Each formula with an instance of a schema expression can be reconstructed to have the expression in terms of a membership relation, which can then be converted into a substituted schema predicate.

Elimination of schema operators. All instances of schema constructs as predicates can be decomposed into predicates containing only schema references. By Theorem 5.6 and by Theorem 5.7 we can remove all instances of schema propositional operators and schema quantification respectively.

Conversion into expressions. Every schema reference used as a predicate can be replaced with the membership relation relating the theta term to the schema reference. The theta term can be replaced by the binding extension.

The resulting formula is one in which there are no instances of schemas used as predicates or as declarations. The only schema construct used is the simple schema product. Hence we have eliminated all forms of schemas and are left with labelled products for which we have a complete set of rules relative to those for Cartesian products.□

6. Related Work

In order to demonstrate the soundness of the logic we have referred to in the preceding sections, a model for Z is required. In this section we discuss briefly some of the models which have been proposed. We also survey some of the other published work in logical treatments of Z schemas.

6.1. SEMANTICS FOR SCHEMAS

DENOTATIONAL MODEL

Spivey (1988) gave the first comprehensive treatment of Z semantics. In developing logics for Z, others found this semantic description hard to use, and proposed a simpler model (Gardiner *et al.*, 1991). The Z Standard has adopted a model based upon that one, and this is used by Brien (1998) in proving the soundness of the logic presented in his thesis.

This model for Z can be given using naive set theory, with the addition of *labelled products* to model schemas as types/bindings. The central notion of the model is that of an *environment*, a mapping of names to values in some suitable universe. A Z specification (or a fragment of a specification; a collection of paragraphs, or a single schema declaration) is then a relation on environments. A predicate is denoted by a set of environments (those in which it holds), and an expression by a mapping from environments to elements of the chosen universe.

Since schemas can be used in a number of different ways, they have a denotation of their own. A schema is a function from environments to sets of bindings—i.e. in any given context, it denotes a particular set of bindings. From this construction, the meaning of a schema predicate in any given environment can be calculated. Likewise, a schema used as an expression denotes the set of bindings (equivalently, environments) gained by applying its meaning function to the current environment. Finally, a schema as a declaration denotes the relation formed in the obvious way from the schema's meaning function.

In this way, the model takes account of the context of each schema use, and this is reflected in the logic we have outlined.

The traditional approach to defining schemas has been by informally defining their properties, but at a semi-formal level, by means of an expansion. The "expansion" of schemas has been defined for those in normal form. This approach has been supported by a form of normalization whereby each schema could be expressed by a normalized declaration and a predicate. Such an approach was an informal way of eliminating instances of schemas from specifications and proofs.

SEMANTICS OF STATE-BASED SYSTEMS

Most attempts to provide a semantic account of the Z state-based system conventions described above will naturally build on the semantics and logic given here. That semantics is not, however, sufficient alone to give an account of this style for using Z since, as we have remarked, in this style different schemas denote different sorts of artifacts.

The account of refinement given in the standard texts (e.g. Spivey, 1992b) amounts to an approach to an axiomatic semantics for these Z state machines. In a different context, having identified schemas with sets of bindings, we might have defined refinement using a notion of inclusion (equivalently, implication). Such a notion of refinement is not appropriate here because whilst it does reduce nondeterminism, it allows the strengthening of the precondition of the operation.

Z state machines may be compared with *action systems* (Back and Sere, 1991). These are similar in structure and principle. It is important to note, however, that the alternatives in an action system are presented with *guards*, and the choice among the actions

is an internal choice. In the usual interpretation of a Z state machine, the actions (operations) have *preconditions*, but no guards. The choice of order of operations is assumed to be external. The reason that Z operation schema preconditions are not interpreted as guards or "firing conditions" is that such a view is incompatible with the Z data refinement rules; the weakening of a precondition would increase nondeterminism.

6.2. OTHER RELATED WORK

Spivey's (1988) account of Z semantics includes ideas for a logic for Z (the schema calculus in particular), but these have not been developed or exploited. A step towards the development of $\mathcal{V}$ was $\mathcal{W}$ (Woodcock and Brien, 1992). That logic has many of the features of $\mathcal{V}$ as presented here, but by failing to take account of context in the calculation of free variables and substitutions, places very heavy constraints on the use of names.

Henson and Reeves (1998) present a logic for Z, together with metalogical results on a similar theme to those presented here. Their logic is rather different, and does not follow the $\mathcal{W}/\mathcal{V}$ approach to substitution. First, a core specification logic Z_C is defined, which uses a classical notion of substitution, and then the more unusual schema operators are defined in terms of Z_C. They argue that this achieves a better separation of concerns than $\mathcal{V}$, which treats the whole of Z at once. The paper cited uses its own semantic model for Z_C, whereas the soundness of $\mathcal{V}$ shown in (Brien, 1998) uses the same semantic model as the Standard.

Though not using Z schemas explicitly, the work of Hoare and He (1998) has many similarities with the treatment of schemas described here. Each predicate in their theory has an associated *alphabet*, and operators are defined to combine such combinations of predicates and alphabets in much the same way as Z schema calculus.

Several projects have implemented proof tools for various logics for Z. Some of these are more evidently logics for Z than others; that is, some take care to support the Standard schema semantics, others do not. These various approaches are surveyed by Martin (1997).

7. Conclusion

In this paper we have demonstrated how schemas are used for a wide variety of specification tasks in Z. These uses may be classified into uses of schemas as expressions, declarations, and predicates. We have demonstrated parts of a logical system which is consistent with the Draft Z Standard and the good texts on Z. It provides a sound means of reasoning about schemas in fullest generality, and one which is amenable to mechanization. Using this calculus, we have shown that specifications using schemas can be re-cast without schemas.

This metalogical result is not necessarily a surprise. Since Z is a first-order theory, it is to be expected that it can be interpreted in ordinary logic and set theory. The complexity of the construction, however, serves to illustrate that the schema structuring notions used in Z add a nontrivial layer of structure. This may help to account for the popularity and perceived utility of Z. Though the ability to remove and replace schema references is unlikely to be of great value in a general-purpose approach to proof in Z (but see Z/EVES, Saaltink, 1997), we have demonstrated elsewhere the practical utility of this logic for reasoning about Z specifications (Brien and Martin, 1995).

We have also observed that most users of Z follow a collection of structuring conventions

in their uses of schemas so that they describe a model of a state-machine. A model for this use of Z sits at a higher level than the basic model and logic for schemas which we have described. This use of schemas—as schematic descriptions of state and operations—is perhaps closer to the use of the term "schema" in other branches of computing science.

Our main future goal in this area is to produce a reference implementation of the $\mathcal{V}$ logic for Standard Z. We also hope to publish a machine-checked proof of soundness for the logic.

Acknowledgements

We are grateful to Michael Butler, Andy Gravell, Martin Henson, Tony Hoare, Ray Turner, and the anonymous referees for comments on this work. Andrew Martin's contribution was written whilst he was employed at the University of Southampton.

References

Abrial, J.-R. (1996). *The B-Book: Assigning Programs to Meanings*. Cambridge, Cambridge University Press.

Back, R.-J., Sere, K. (1991). Stepwise refinement of action systems. *Struct. Program.*, **12**, 17–30.

Bowen, J. P. (1996). Z archive. `URL:http://www.comlab.ox.ac.uk/archive/z.html`.

Bowen, J. P., Hinchey, M. G., Till, D., eds (1997). In *ZUM'97: The Z formal specification notation, Proceedings of the 10th International Conference of Z Users, Reading, UK, April 1997,* LNCS **1212**, Berlin, Heidelberg, Springer.

Brien, S. M. (1994). The Development of Z. In Andrews, D. J., Groote, J. F. and Middelburg, C. A., eds, *Semantics of Specification Languages (SoSL), Workshops in Computing*, pp. 1–14. Springer.

Brien, S. M. (1998). A logic and model for the Z standard. D.Phil. Thesis, University of Oxford.

Brien, S. M., Martin, A. P. (1995). A tutorial on proof in Standard Z. Technical Monograph PRG-120, Programming Research Group, Oxford University Computing Laboratory, Wolfson Building, Parks Road, Oxford, OX1 3QD, UK. Presented at ZUM'95.

Calvacanti, A. (1997). A refinement calculus for Z. D.Phil. Thesis, University of Oxford. Available as Technical Monograph PRG-125, Oxford University Computing Laboratory.

Fidge, C. J., Kearney, P., Martin, A. P. (1998). Applying the Cogito program development environment to real-time system design. In McDonald, C., ed., *Computer Science '98*, pp. 367–378. Springer. *Proceedings of the 21st Australasian Computer Science Conference, Perth, 4–6 February 1998.* Australian Computer Science Communications, volume 20, no. 1. Also available as Technical Report SVRC-TR-97-36.

Gardiner, P. H. B., Lupton, P. J., Woodcock, J. C. P. (1991). A simpler semantics for Z. In Nicholls, J. E., ed., *Z User Workshop, Oxford 1990, Workshops in Computing*, pp. 3–11. Springer.

Henson, M. C., Reeves, S. (1998). Investigating Z. Technical Report CSM-317, Department of Computer Science, University of Essex. *Journal of Logic and Computation*, to appear.

Hoare, C. A. R., He, J. (1998). *Unifying Theories of Programming*. Series in Computer Science, Prentice Hall.

Martin, A. (1997). Why effective proof tool support for Z is hard. Technical Report 97-34, Software Verification Research Centre, School of Information Technology, The University of Queensland, Brisbane 4072, Australia.

Nicholls, J., ed. (1995). *Z Notation*. Z Standards Panel, ISO Panel JTC1/SC22/WG19 (Rapporteur Group for Z). Version 1.2, ISO Committee Draft; CD 13568.

Saaltink, M. (1997). The Z/EVES system. In Bowen *et al.* (1997), pp. 72–85.

Spivey, J. M. (1988). *Understanding Z: A Specification Language and its Formal Semantics,* volume 3 *of Cambridge Tracts in Theoretical Computer Science*, Cambridge University Press.

Spivey, J. M. (1992a). *The* fUZZ *Manual,* 2nd edn, Computing Science Consultancy, 34 Westlands Grove, Stockton Lane, York YO3 0EF, UK.

Spivey, J. M. (1992b). *The Z Notation: A Reference Manual,* 2nd edn. Prentice-Hall.

Spivey, J. M., Sufrin, B. A. (1990). Type inference in Z. In Bjørner, D., Hoare, C. A. R. and Langmaack, H., eds, *VDM'90: VDM and Z—Formal Methods in Software Development,* LNCS **428**. pp. 426–451. Springer.

Stepney, S., Barden, R., Cooper, D., eds. (1992). *Object Orientation in Z, Workshops in Computing*. Springer.

Stoddart, B. (1997). An introduction to the event calculus. In Bowen *et al.* (1997), pp. 10–34.

Woodcock, J. C. P., Brien, S. M. (1992). $\mathcal{W}$: A logic for Z. In *Proceedings of the Sixth Z User Meeting*. Springer.
Woodcock, J. C. P., Davies, J. (1996). *Using Z: Specification, Refinement, and Proof*. Europe, Prentice-Hall.
Wordsworth, J. B. (1993). *Software Development with Z: A Practical Approach to Formal Methods in Software Engineering*. Addison-Wesley Publishing Company.

Appendix: The $\mathcal{V}$ logic for Z

This material is presented with commentary in Brien (1998).

Notational Conventions

We use the following meta-variable conventions:

Predicates	$\mathsf{P}, \mathsf{Q}, \mathsf{R}$	Variables	$\mathsf{x}, \mathsf{y}, \mathsf{z}$
Expressions	e, u	Sets	s, t
Binding	b	Schemas	S, T
Specifications	Γ		

A.1. STRUCTURAL RULES

$$\frac{\Gamma \mid \mathsf{P} \vdash \mathsf{Q} \qquad \Gamma \vdash \mathsf{P} \qquad \Gamma \mid \mathsf{P} \vdash \mathsf{Q} \checkmark}{\Gamma \vdash \mathsf{Q}} \; (\mathit{cut}\ \mathsf{P})$$

$$\frac{}{\Gamma \mid \mathsf{P} \vdash \mathsf{P}} \; \mathit{Assum} \qquad \frac{\Gamma \vdash \mathsf{Q}}{\Gamma \mid \mathsf{P} \vdash \mathsf{Q}} \; (\mathit{thin}\ \vdash)$$

A.2. PROPOSITIONS

$$\begin{aligned} \mathbf{true} &\equiv \neg\ \mathbf{false} \\ \neg\ \mathsf{P} &\equiv \mathsf{P} \Rightarrow \mathbf{false} \\ \mathsf{P} \Leftrightarrow \mathsf{Q} &\equiv (\mathsf{P} \Rightarrow \mathsf{Q}) \quad \wedge \quad (\mathsf{Q} \Rightarrow \mathsf{P}) \end{aligned}$$

$$\frac{}{\Gamma \mid \mathbf{false} \vdash \mathsf{P}} \; (\mathbf{false}\ \vdash) \qquad \frac{\Gamma \mid \mathsf{P} \Rightarrow \mathbf{false} \vdash \mathsf{P}}{\Gamma \vdash \mathsf{P}} \; (\Rightarrow \mathbf{false})$$

$$\frac{\Gamma \mid \mathsf{P} \mid \mathsf{Q} \vdash \mathsf{R}}{\Gamma \mid \mathsf{P} \wedge \mathsf{Q} \vdash \mathsf{R}} \; (\wedge\vdash) \qquad \frac{\Gamma \vdash \mathsf{P} \qquad \Gamma \vdash \mathsf{Q}}{\Gamma \vdash \mathsf{P} \wedge \mathsf{Q}} \; (\vdash\wedge)$$

$$\frac{\Gamma \mid \mathsf{P} \vdash \mathsf{R} \quad \Gamma \mid \mathsf{Q} \vdash \mathsf{R}}{\Gamma \mid \mathsf{P} \vee \mathsf{Q} \vdash \mathsf{R}} \; (\vee\vdash) \qquad \frac{\Gamma \vdash \mathsf{P}}{\Gamma \vdash \mathsf{P} \vee \mathsf{Q}} \; (\vdash\vee) \qquad \frac{\Gamma \vdash \mathsf{Q}}{\Gamma \vdash \mathsf{P} \vee \mathsf{Q}} \; (\vdash\vee)$$

$$\frac{\Gamma \vdash \mathsf{P} \quad \Gamma \mid \mathsf{Q} \vdash \mathsf{R}}{\Gamma \mid \mathsf{P} \Rightarrow \mathsf{Q} \vdash \mathsf{R}} \; (\Rightarrow\vdash) \qquad \frac{\Gamma \mid \mathsf{P} \vdash \mathsf{Q}}{\Gamma \vdash \mathsf{P} \Rightarrow \mathsf{Q}} \; (\vdash\Rightarrow)$$

$$\frac{}{\Gamma \vdash \langle\!|x{\rightsquigarrow}e|\!\rangle_{\odot}(P \Rightarrow Q) \quad \Leftrightarrow \quad (\langle\!|x{\rightsquigarrow}e|\!\rangle_{\odot}P) \Rightarrow (\langle\!|x{\rightsquigarrow}e|\!\rangle_{\odot}Q)}\ (\odot\Rightarrow)$$

A.3. QUANTIFICATION

$$\frac{\Gamma \mid \langle\!|x{\rightsquigarrow}y|\!\rangle_{\odot}P \vdash Q}{\Gamma \mid \forall[x] \bullet P \vdash Q}\ (\forall\vdash) \qquad \frac{\Gamma \mid (e \in s) \ \Rightarrow\ \langle\!|x{\rightsquigarrow}e|\!\rangle_{\odot}P \vdash Q \qquad \Gamma \vdash e \in s\ \checkmark}{\Gamma \mid \forall x : s \bullet P \vdash Q}\ (\forall\vdash)$$

$$\frac{\Gamma \vdash P \qquad \vdash \text{`x'} \setminus [\Gamma]}{\Gamma \vdash \forall[x] \bullet P}\ (\forall\vdash) \qquad \frac{\Gamma \vdash x \in s \ \Rightarrow\ P \qquad \vdash \text{`x'} \setminus [\Gamma] \qquad \Gamma \vdash \text{`x'} \setminus s}{\Gamma \vdash \forall x : s \bullet P}\ (\vdash\forall)$$

$$\frac{}{\Gamma \vdash \langle\!|x{\rightsquigarrow}e|\!\rangle_{\odot} \forall x : s \bullet P \Leftrightarrow \forall x : \langle\!|x{\rightsquigarrow}e|\!\rangle_{\odot}s \bullet P}\ (\odot\forall)$$

$$\frac{\Gamma \vdash \text{`y'} \setminus x \qquad \Gamma \vdash \text{`y'} \setminus e}{\Gamma \vdash \langle\!|x{\rightsquigarrow}e|\!\rangle_{\odot} \forall y : s \bullet P \Leftrightarrow \forall y : \langle\!|x{\rightsquigarrow}e|\!\rangle_{\odot}s \bullet \langle\!|x{\rightsquigarrow}e|\!\rangle_{\odot}P}\ (\odot\forall)$$

$$\frac{\Gamma \vdash \text{`y'} \setminus x \qquad \Gamma \vdash \text{`y'} \setminus e}{\Gamma \vdash \langle\!|x{\rightsquigarrow}e|\!\rangle_{\odot} \forall[y] \bullet P \Leftrightarrow \forall[y] \bullet \langle\!|x{\rightsquigarrow}e|\!\rangle_{\odot}P}\ (\odot\forall)$$

$$\frac{\Gamma \vdash \text{`b'} \setminus \forall S \bullet P}{\Gamma \vdash \forall S \bullet P \quad \Leftrightarrow \quad \forall b : S \bullet b_{\odot}P}\ :[]$$

$$\exists[x] \bullet P \equiv \neg\ \forall[x] \bullet \neg\ P$$
$$\exists S \bullet P \equiv \neg\ \forall S \bullet \neg\ P$$

A.4. SUBSTITUTION

$$\frac{}{\Gamma \vdash \langle\!|x{\rightsquigarrow}x|\!\rangle_{\odot}P \Leftrightarrow P}\ (\odot1) \qquad \frac{\Gamma \vdash \text{`x'} \setminus P}{\Gamma \vdash \langle\!|x{\rightsquigarrow}e|\!\rangle_{\odot}P \Leftrightarrow P}\ (\odot2)$$

$$\frac{\Gamma \mid \langle\!|x{\rightsquigarrow}e|\!\rangle_{\odot}b \vdash \text{`x'} \setminus P}{\Gamma \vdash \langle\!|x{\rightsquigarrow}e|\!\rangle_{\odot}(b_{\odot}P) \quad \Leftrightarrow \quad (\langle\!|x{\rightsquigarrow}e|\!\rangle_{\odot}b)_{\odot}P}\ (\odot3)$$

$$\frac{\Gamma \vdash \text{`x'} \setminus y \qquad \Gamma \vdash \text{`x'} \setminus u}{\Gamma \vdash \langle\!|y{\rightsquigarrow}u|\!\rangle_{\odot}\left(\langle\!|x{\rightsquigarrow}e|\!\rangle_{\odot}P\right) \quad \Leftrightarrow \quad \langle\!|x{\rightsquigarrow}\langle\!|y{\rightsquigarrow}u|\!\rangle_{\odot}e|\!\rangle_{\odot}\left(\langle\!|y{\rightsquigarrow}u|\!\rangle_{\odot}P\right)}\ (\odot4)$$

$$\frac{\Gamma \vdash \text{`x'} \setminus v \qquad \cdots \qquad \Gamma \vdash \text{`y'} \setminus v}{\Gamma \vdash \langle\!|x{\rightsquigarrow}e, \ldots, y{\rightsquigarrow}u, z{\rightsquigarrow}v|\!\rangle_{\odot}P \quad \Leftrightarrow \quad \langle\!|x{\rightsquigarrow}e, \ldots, y{\rightsquigarrow}u|\!\rangle_{\odot}\left(\langle\!|z{\rightsquigarrow}v|\!\rangle_{\odot}P\right)}\ (\odot5)$$

A.5. DECLARATIONS

$$\frac{\Gamma \vdash \forall[x] \bullet P}{\Gamma \mid [x] \vdash P}\ ([]\vdash) \qquad \frac{\Gamma \mid [x] \vdash P \qquad \vdash \Gamma \mid [x]\ \surd}{\Gamma \vdash \forall[x] \bullet P}\ (\vdash []) \qquad \frac{\Gamma \vdash \forall S \bullet P}{\Gamma \mid S \vdash P}\ (:\vdash) \qquad \frac{\Gamma \mid S \vdash P}{\Gamma \vdash \forall S \bullet P}\ (\vdash :)$$

$$\frac{\Gamma \vdash (\!| x \rightsquigarrow e |\!)_\odot P}{\Gamma \mid x := e \vdash P}\ (:=\vdash) \qquad \frac{\Gamma \mid x := e \vdash P}{\Gamma \vdash (\!| x \rightsquigarrow e |\!)_\odot P}\ (\vdash \rightsquigarrow)$$

A.6. EQUALITY AND MEMBERSHIP

$$\frac{}{\Gamma \vdash e = e}\ \mathit{Refl} \qquad \frac{\Gamma \vdash (\!| x \rightsquigarrow u |\!)_\odot P \qquad \Gamma \vdash e = u}{\Gamma \vdash (\!| x \rightsquigarrow e |\!)_\odot P}\ \mathit{Leib}$$

$$\frac{\Gamma \vdash `x' \setminus t \qquad \Gamma \vdash `x' \setminus s}{\Gamma \vdash (\forall x : s \bullet x \in t \quad \wedge \quad \forall x : t \bullet x \in s) \Leftrightarrow s = t}\ \mathit{Extn}$$

$$\frac{\Gamma \vdash `x' \setminus s}{\Gamma \vdash (\!| x \rightsquigarrow y |\!)_\odot (e \in s) \quad \Leftrightarrow \quad ((\!| x \rightsquigarrow y |\!)_\odot e) \in s}\ (\in \odot) \qquad \frac{\Gamma \vdash `x' \setminus e}{\Gamma \vdash (\!| x \rightsquigarrow y |\!)_\odot (e \in s) \quad \Leftrightarrow \quad e \in (\!| x \rightsquigarrow y |\!)_\odot s}\ (\odot \in)$$

$$\frac{\Gamma \vdash `x' \setminus u}{\Gamma \vdash (\!| x \rightsquigarrow y |\!)_\odot (e = u) \quad \Leftrightarrow \quad ((\!| x \rightsquigarrow y |\!)_\odot e) = u}\ (\odot =)$$

A.7. SET THEORY

$$\frac{}{\Gamma \vdash e \in \{u_1, u_2, \ldots\} \quad \Leftrightarrow (e = u_1 \vee e \in u_2 \ldots)}\ \mathit{Pairing}$$

$$\frac{\Gamma \vdash `x' \setminus e}{\Gamma \vdash e \in \bigcup s \quad \Leftrightarrow \quad (\exists x : s \bullet e \in x)}\ \mathit{Union}$$

$$\frac{\Gamma \vdash `x' \setminus s}{\Gamma \vdash e \in \mathbb{P} s \quad \Leftrightarrow \quad (\forall x : e \bullet x \in s)}\ \mathit{Powerset}$$

$$\frac{\Gamma \vdash S \setminus e}{\Gamma \vdash e \in \{S \bullet u\} \quad \Leftrightarrow \quad \exists S \bullet e = u}\ \mathit{Comp}$$

$$\frac{}{\vdash \exists[x] \bullet \forall y : x \bullet \mathbf{false}}\ \mathit{Empty}$$

$$\frac{}{\vdash \exists[x] \bullet \exists 0 : x \bullet \exists\, \mathit{succ} : x \rightarrowtail\!\!\!\!\!\rightarrow (x \setminus \{0\}) \bullet \mathbf{true}}\ \mathit{Infty}$$

$$\frac{}{\vdash \exists[\mathcal{W}] \bullet \ \cdots}\ \mathit{WRep}$$

$$\frac{}{\Gamma \vdash \exists x : s \rightarrow\!\!\!\!\!\rightarrow t \bullet \mathbf{true} \Rightarrow \exists y : t \rightarrowtail s \bullet \mathbf{true}}\ \mathit{Choice}$$

A.8. TUPLES

$$\frac{}{\Gamma \vdash \mathsf{x} = (\mathsf{e}_1, \ldots, \mathsf{e}_n) \quad \Leftrightarrow \quad \mathsf{x}.1 = \mathsf{e}_1 \land \cdots \land \mathsf{x}.n = \mathsf{e}_n} \; \textit{Tuple}$$

$$\frac{}{\Gamma \vdash (\mathsf{e}_1, \ldots, \mathsf{e}_n).i \quad = \quad \mathsf{e}_i \qquad (1 \leq i \leq n)} \; \textit{TupleSel}$$

$$\frac{}{\Gamma \vdash \mathsf{e} \in \mathsf{s}_1 \times \cdots \times \mathsf{s}_n \quad \Leftrightarrow \quad \mathsf{e}_1 \in \mathsf{s}_1 \land \cdots \land \mathsf{e}_n \in \mathsf{s}_n} \; \textit{CartProd}$$

$$\frac{\Gamma \vdash \exists_1 \mathsf{x} : \mathsf{f} \bullet \mathsf{x}.1 = \mathsf{e} \qquad \Gamma \vdash \text{`x'} \setminus \mathsf{e}}{\Gamma \vdash \mathsf{y} = \mathsf{f}(\mathsf{e}) \quad \Leftrightarrow \quad (\mathsf{e}, \mathsf{y}) \in \mathsf{f}} \; \textit{Funct}$$

$$\frac{\Gamma \vdash \mathsf{b} \setminus \mathsf{x}}{\Gamma \vdash \mathsf{b}_\odot(\mathsf{f}(\mathsf{x})) \quad = \quad (\mathsf{b}_\odot \mathsf{f})\mathsf{x}} \; \textit{SubFun} \qquad \frac{\Gamma \vdash \mathsf{b} \setminus \mathsf{f}}{\Gamma \vdash \mathsf{b}_\odot(\mathsf{f}(\mathsf{x})) \quad = \quad \mathsf{f}(\mathsf{b}_\odot \mathsf{x})} \; \textit{SubArg}$$

$$\frac{\Gamma \mid \mathsf{S} \vdash \text{`x'} \setminus \mathsf{e} \qquad \Gamma \mid \mathsf{S} \vdash \text{`y'} \setminus \mathsf{e}}{\Gamma \vdash \mu \mathsf{S} \bullet \mathsf{e} \quad = \quad \left\{ \mathsf{y} : \mathbb{P} \widehat{\mathsf{S}}; \; \mathsf{x} : \{ \; \widehat{\mathsf{S}} \bullet \mathsf{e} \; \} \mid \mathsf{x} \in \{ \; \mathsf{y} \bullet \mathsf{e} \; \} \quad \bullet \quad (\mathsf{y}, \mathsf{x}) \right\} \; (\mathsf{S})} \; \textit{Desc}$$

A.9. BINDINGS

$$\frac{}{(\!| \mathsf{x}_1 \rightsquigarrow \mathsf{e}_1, \ldots, \mathsf{x}_n \rightsquigarrow \mathsf{e}_n |\!).\mathsf{x}_i = \mathsf{e}_i \qquad (1 \leq i \leq n)} \; \textit{Bind}$$

$$\frac{}{\Gamma \vdash \mathsf{b} = (\!| \mathsf{x}_1 \rightsquigarrow \mathsf{e}_1, \ldots, \mathsf{x}_n \rightsquigarrow \mathsf{e}_n |\!) \quad \Leftrightarrow \quad \mathsf{b}.\mathsf{x}_1 = \mathsf{e}_1 \land \cdots \land \mathsf{b}.\mathsf{x}_n = \mathsf{e}_n} \; \textit{BindSel}$$

$$\frac{}{\Gamma \vdash \mathsf{b} \in [\mathsf{x}_1 : \mathsf{s}_1; \; \cdots; \; \mathsf{x}_n : \mathsf{s}_n] \quad \Leftrightarrow \quad \mathsf{b}.\mathsf{x}_1 \in \mathsf{s}_1 \land \cdots \land \mathsf{b}.\mathsf{x}_n \in \mathsf{s}_n} \; \textit{SchProd}$$

$$\frac{}{\Gamma \vdash \theta \mathsf{S}.\mathsf{x} \quad = \quad \mathsf{x}} \; \textit{Theta}$$

$$\frac{}{\Gamma \vdash (\!| \mathsf{x}_1 \rightsquigarrow \mathsf{e}_1, \ldots, \mathsf{x}_n \rightsquigarrow \mathsf{e}_n |\!)[\mathsf{x}_1/\mathsf{y}] \quad = \quad (\!| \mathsf{y} \rightsquigarrow \mathsf{e}_1, \ldots, \mathsf{x}_n \rightsquigarrow \mathsf{e}_n |\!)} \; (\!|/|\!)$$

$$\frac{}{\Gamma \vdash \mathsf{b} \in \mathsf{S}[\mathsf{x}/\mathsf{y}] \quad \Leftrightarrow \quad \mathsf{b}[\mathsf{x}/\mathsf{y}] \in \mathsf{S}} \; [/]$$

$$\frac{}{\Gamma \vdash (\mathsf{b})'.\mathsf{x}' \quad = \quad \mathsf{b}.\mathsf{x}} \; \textit{Dec} \qquad \frac{}{\Gamma \vdash (\mathsf{b})' \in (\mathsf{S})' \quad \Leftrightarrow \quad \mathsf{b} \in \mathsf{S}} \; \textit{Dec}$$

A.10. SCHEMAS

$$\begin{aligned} \mathsf{S}; \; \mathsf{T} &\equiv \mathsf{S} \land \mathsf{S} \\ \mathsf{S} \Rightarrow \mathsf{T} &\equiv \neg \; \mathsf{S} \;\; \lor \;\; \mathsf{T} \\ \mathsf{S} \Leftrightarrow \mathsf{T} &\equiv (\mathsf{S} \Rightarrow \mathsf{T}) \quad \land \quad (\mathsf{S} \Rightarrow \mathsf{T}) \\ \exists \mathsf{S} \bullet \mathsf{T} &\equiv \neg \; \forall \mathsf{S} \bullet \neg \; \mathsf{T} \end{aligned}$$

$$\frac{}{\Gamma \vdash \theta \mathsf{S} \in \mathsf{S} \quad \Leftrightarrow \quad [\mathsf{S}]} \; \theta[]$$

$$\frac{}{\Gamma \vdash [S \mid P] \quad \Leftrightarrow \quad [S] \quad \wedge \quad P}\ [|] \qquad \frac{}{\Gamma \vdash [S \wedge T] \quad \Leftrightarrow \quad [S] \quad \wedge \quad [T]}\ [\wedge]$$

$$\frac{}{\Gamma \vdash [\neg S] \quad \Leftrightarrow \quad \neg [S]}\ [\neg] \qquad \frac{}{\Gamma \vdash [S \Rightarrow T] \quad \Leftrightarrow \quad [S] \quad \Rightarrow \quad [T]}\ [\Rightarrow]$$

$$\frac{\Gamma \vdash S \setminus T}{\Gamma \vdash [\forall S \bullet T] \ \Leftrightarrow \ \forall S \bullet [T]}\ [\forall] \qquad \frac{}{\Gamma \vdash [\forall S[x/y] \bullet T[x/y]] \ \Leftrightarrow \ [\forall S \bullet T]}\ [\forall[/]]$$

A.11. GENERIC DEFINITIONS

The following rules for generic definitions are valid only when the names of the generic variables/schemas are distinct:

$$y[x] := e \qquad \equiv \qquad \forall [x] \bullet y_{[x]} \ = \ e$$

$$[x]S \qquad \equiv \qquad \forall [x] \bullet [(S)_{[x]}]$$

A.12. FREE VARIABLES

The following rules give a context dependent definition of the free variables of formulae in $\mathcal{V}$. For a set of rules to derive the type of these formulae see the Draft Standard (Nicholls, 1995). In the following we assume that the meta-variables x and y are not instantiated with the same variable.

PREDICATES

$$\frac{}{\Gamma \vdash \text{`x'} \setminus y} \qquad \frac{}{\Gamma \vdash \text{`x'} \setminus \mathbf{false}}$$

$$\frac{\Gamma \vdash \text{`x'} \setminus e \qquad \Gamma \vdash \text{`x'} \setminus s}{\Gamma \vdash \text{`x'} \setminus e \in s} \qquad \frac{\Gamma \vdash \text{`x'} \setminus e \qquad \Gamma \vdash \text{`x'} \setminus u}{\Gamma \vdash \text{`x'} \setminus e = u}$$

$$\frac{\Gamma \vdash \text{`x'} \setminus P \qquad \Gamma \vdash \text{`x'} \setminus Q}{\Gamma \vdash \text{`x'} \setminus P \wedge Q} \qquad \frac{\Gamma \vdash \text{`x'} \setminus P \qquad \Gamma \vdash \text{`x'} \setminus Q}{\Gamma \vdash \text{`x'} \setminus P \Rightarrow Q}$$

$$\frac{\Gamma \vdash \text{`x'} \setminus S \qquad \Gamma \mid S \vdash \text{`x'} \setminus P}{\Gamma \vdash \text{`x'} \setminus \forall S \bullet P} \qquad \frac{\Gamma \vdash \text{`x'} \setminus S \qquad \Gamma \vdash S \ succ \ x}{\Gamma \vdash \text{`x'} \setminus \forall S \bullet P}$$

$$\frac{\Gamma \vdash \text{`x'} \setminus e}{\Gamma \vdash \text{`x'} \setminus (\!| x \rightsquigarrow e |\!)_{\odot} P} \qquad \frac{\Gamma \vdash \text{`x'} \setminus e \qquad \Gamma \mid y := e \vdash \text{`x'} \setminus P}{\Gamma \vdash \text{`x'} \setminus (\!| y \rightsquigarrow e |\!)_{\odot} P}$$

$$\frac{\Gamma \mid [y] \vdash \text{`x'} \setminus P}{\Gamma \vdash \text{`x'} \setminus \forall [y] \bullet P} \qquad \frac{}{\Gamma \vdash \text{`x'} \setminus \forall [x] \bullet P}$$

EXPRESSIONS

$$\frac{\Gamma \vdash \text{`x'} \setminus \mathsf{e}_1 \qquad \cdots \qquad \Gamma \vdash \text{`x'} \setminus \mathsf{e}_n}{\Gamma \vdash \text{`x'} \setminus \{\mathsf{e}_1, \ldots, \mathsf{e}_n\}} \qquad \frac{\Gamma \vdash \text{`x'} \setminus \mathsf{s} \quad \Gamma \vdash \text{`x'} \setminus \mathsf{P} \quad \text{`}\Gamma \vdash \mathsf{x}\text{'} \setminus \mathsf{e}}{\Gamma \vdash \text{`x'} \setminus \{\mathsf{y} : \mathsf{s} \mid \mathsf{P} \bullet \mathsf{e}\}}$$

$$\frac{\Gamma \vdash \text{`x'} \setminus \mathsf{s}}{\Gamma \vdash \text{`x'} \setminus \{\mathsf{x} : \mathsf{s} \mid \mathsf{P} \bullet \mathsf{e}\}} \qquad \frac{\Gamma \vdash \text{`x'} \setminus \mathsf{s}}{\Gamma \vdash \text{`x'} \setminus \mathbb{P}\mathsf{s}}$$

$$\frac{\Gamma \vdash \text{`x'} \setminus \mathsf{s}}{\Gamma \vdash \text{`x'} \setminus \bigcup \mathsf{s}} \qquad \frac{\Gamma \vdash \text{`x'} \setminus \mathsf{e}_1 \qquad \cdots \qquad \Gamma \vdash \text{`x'} \setminus \mathsf{e}_n}{\Gamma \vdash \text{`x'} \setminus (\mathsf{e}_1, \ldots, \mathsf{e}_n)}$$

$$\frac{\Gamma \vdash \text{`x'} \setminus \mathsf{e}}{\Gamma \vdash \text{`x'} \setminus \mathsf{e}.i} \qquad \frac{\Gamma \vdash \text{`x'} \setminus \mathsf{s}_1 \quad \cdots \quad \Gamma \vdash \text{`x'} \setminus \mathsf{s}_n}{\Gamma \vdash \text{`x'} \setminus \mathsf{s}_1 \times \cdots \times \mathsf{s}_n}$$

$$\frac{\Gamma \vdash \text{`x'} \setminus \mathsf{S} \qquad \Gamma \vdash \mathsf{S} \setminus \mathsf{x}}{\Gamma \vdash \text{`x'} \setminus \theta\mathsf{S}}$$

$$\frac{\Gamma \vdash \text{`x'} \setminus \mathsf{e}_1 \qquad \cdots \qquad \Gamma \vdash \text{`x'} \setminus \mathsf{e}_n}{\Gamma\text{`x'} \setminus (\!|\mathsf{x}_1 \rightsquigarrow \mathsf{e}_1, \ldots, \mathsf{x}_n \rightsquigarrow \mathsf{e}_n|\!)} \qquad \frac{\Gamma \vdash \text{`x'} \setminus \mathsf{b}}{\Gamma \vdash \text{`x'} \setminus \mathsf{b}.\mathsf{y}}$$

$$\frac{\Gamma \vdash \text{`x'} \setminus \mathsf{b}}{\Gamma \vdash \text{`x'} \setminus \mathsf{b}[\mathsf{x}/\mathsf{y}]}$$

SCHEMAS

$$\frac{\Gamma \vdash \text{`x'} \setminus \mathsf{S} \qquad \Gamma \vdash \text{`x'} \setminus \mathsf{T}}{\Gamma \vdash \text{`x'} \setminus \mathsf{S} \land \mathsf{T}} \qquad \frac{\Gamma \vdash \text{`x'} \setminus \mathsf{S} \qquad \Gamma \vdash \text{`x'} \setminus \mathsf{T}}{\Gamma \vdash \text{`x'} \setminus \mathsf{S} \Rightarrow \mathsf{T}}$$

$$\frac{\Gamma \vdash\vdash \text{`x'} \setminus \mathsf{S} \qquad \Gamma \vdash \text{`x'} \setminus \mathsf{T}}{\Gamma \vdash \text{`x'} \setminus \forall \mathsf{S} \bullet \mathsf{T}}$$

$$\frac{\Gamma \vdash \text{`x'} \setminus \mathsf{S} \qquad \Gamma \mid \mathsf{S} \vdash \text{`x'} \setminus \mathsf{P}}{\Gamma \vdash \text{`x'} \setminus (\mathsf{S} \mid \mathsf{P})} \qquad \frac{\Gamma \vdash \text{`x'} \setminus \mathsf{S} \qquad \Gamma \vdash \mathsf{S}\ \mathit{succ}\ \mathsf{x}}{\Gamma \vdash \text{`x'} \setminus (\mathsf{S} \mid \mathsf{P})}$$

$$\frac{\Gamma \vdash \mathsf{x} \setminus \mathsf{s}}{\Gamma \vdash \text{`x'} \setminus \mathsf{y} : \mathsf{s}} \qquad \frac{\Gamma \vdash \mathsf{S} \setminus \mathsf{x} \qquad \Gamma \vdash \text{`x'} \setminus \mathsf{S}}{\Gamma \vdash \text{`x'} \setminus [\mathsf{S}]}$$

PARAGRAPHS

$$\frac{\vdash \text{`x'} \setminus [\Gamma] \qquad \Gamma \vdash \text{`x'} \setminus \mathsf{P}}{\text{`x'} \setminus [\Gamma \mid \mathsf{P}]} \qquad \frac{\vdash \text{`x'} \setminus [\Gamma] \qquad \Gamma \vdash \text{`x'} \setminus [\mathsf{S}]}{\vdash \text{`x'} \setminus [\Gamma \mid \mathsf{S}]}$$

COMPONENT NAMES

$$\frac{\Gamma \vdash \text{`}\mathsf{x}_1\text{'} \setminus \mathsf{P} \quad \cdots \quad \Gamma \vdash \text{`}\mathsf{x}_n\text{'} \setminus \mathsf{P} \qquad \Gamma \vdash \mathsf{S} \mathbin{:\!:} \mathcal{P}(\mathsf{x}_1 \rightsquigarrow \tau_1 \mathbin{⨯} \cdots \mathbin{⨯} \mathsf{x}_n \rightsquigarrow \tau_n)}{\Gamma \vdash \mathsf{S} \setminus \mathsf{P}}$$

$$\frac{\Gamma \vdash \mathsf{S} \mathrel{⦂} \mathcal{P}(\mathsf{x}_1 \rightsquigarrow \tau_1 \mathbin{⨭} \cdots \mathbin{⨭} \mathsf{x}_n \rightsquigarrow \tau_n)}{\Gamma \vdash \mathsf{S}\ \mathit{succ}\ \mathsf{x}}$$

$$\frac{\text{`x'} \setminus \mathsf{s}}{\text{`x'} \setminus \mathsf{y} : \mathsf{s}}$$

$$\frac{\Gamma \vdash \text{`}\mathsf{x}_1\text{'} \setminus \mathsf{e} \quad \cdots \quad \Gamma \vdash \text{`}\mathsf{x}_n\text{'} \setminus \mathsf{e} \qquad \Gamma \vdash \mathsf{S} \mathrel{⦂} \mathcal{P}(\mathsf{x}_1 \rightsquigarrow \tau_1 \mathbin{⨭} \cdots \mathbin{⨭} \mathsf{x}_n \rightsquigarrow \tau_n)}{\mathsf{S} \setminus \mathsf{e}}$$

Originally Received xx September 1998
Accepted xx May 1999