Abstract. This paper provides an introduction to the specification language Z from a logical perspective. The possibility of presenting Z in this way is a consequence of a number of joint publications on Z logic that Henson and Reeves have co-written since 1997. We provide an informal as well as formal introduction to Z logic and show how it may be used, and extended, to investigate issues such as equational logic, the logic of preconditions, the issue of monotonicity and both operation and data refinement.

Keywords: Formal methods, specification language, logic, Z
1 INTRODUCTION

This paper describes an approach to Z logic – it is relatively unconcerned with Z semantics, except insofar as the existence of a non-trivial model is useful for establishing the consistency of the logic. The paper neither attempts to replicate, nor to extend, the excellent work on Z standardisation which has led to ISO standard 13568.1 It is, rather, complementary, seeking to explore and express the logical preliminaries of Z2 and aiming to describe those uncontroversial properties of the major elements of the language, in particular, the language of schemas and its calculus.

The approach to Z logic taken here is mainly based on three papers [17, 18, 19]; these remain the comprehensive technical resource for two separate though related approaches (we make some reference to the distinction in Section 3.5). Our objective in this paper is to provide a more accessible overview of that work and to highlight some more advanced related work beyond specification, in particular in the theory of refinement, that becomes possible by virtue of the Z logic that we describe.

The paper is structured in three parts. The first is the least formal and most accessible; it explores initial considerations concerning the formalisation of vernacular3 Z with particular reference to the novel features (those that take Z beyond, at least in expressivity, higher-order logic) concerning schema types and bindings. The second part of the paper is a more formally presented account of Z logic (the logic $Z_C$) and how that may be extended by means of a series of conservative extensions to more comprehensive logical systems with wider coverage. We are by no means encyclopaedic and the earlier papers referred to above contain more detail and a more formal account. The final part of the paper contains the most advanced material: it looks beyond Z as a specification language and $Z_C$ as a logic for reasoning about specification. It demonstrates the further utility of such a logic by showing how various theories of equality, operation and data refinement can be integrated with, and issues such as monotonicity explored within, the base logic in a smooth and systematic manner: something made possible with a logic in place. This survey relies on the reader’s previous general knowledge of the topics it briefly surveys. The paper ends with some concluding remarks, our acknowledgements and relevant references to the literature.

---

1 The Z Standard does not provide a logic. The strategic decision to exclude a logic was reported in [22]. An inconsistency [16] was discovered in the (unfinished) draft logic submitted as part of the ISO Committee Draft 1.2 of the Z Standard in 1995.

2 Although beginning from its logical first principles, this paper does not begin Z itself from first principles. The reader is assumed to be familiar with Z notation and concepts as described in one of the better textbooks, for example, [33].

3 By vernacular Z we refer to Z as it has been used in practice and as it is reported in informal and semi-formal accounts in the literature.
2 INITIAL CONSIDERATIONS

We take it as self-evident that any formal specification should permit precise consequences to be drawn: the emphasis in the term *formal method* should fall on the second word and not the first. A language, even one with a semantics, is impoverished if there is no logic: it would provide no means for drawing those consequences in a methodical, reproducible and agreed fashion. In this first part of the paper we re-introduce the key features of specification in Z from a logical perspective. Our objective is to motivate and introduce the basic principles of the logic $\mathcal{Z}_c$ and to explain why this core logic is a satisfactory basis for establishing logical apparatus for a range of Z concepts.

2.1 Z Schemas and Bindings

At the heart of Z is the *schema*. Schemas are usually used in two ways: for describing the *state space* of a system and for describing *operations* which the system may perform.

**Example 1.** Informal state space: a jug of capacity 250ml of water having a current volume and a current temperature. As a schema:

```
Jug
 volume : N
 temp : N
```

\[
\begin{array}{l}
\text{volume} \leq 250 \\
\text{temp} \leq 100 \\
\end{array}
\]

Written in linear form this would be:

\[
Jug \equiv [\text{volume} : N; \text{temp} : N | \text{volume} \leq 250 \land \text{temp} \leq 100]
\]

This schema has the name *Jug* and introduces two observations, *volume* and *temp*, which have some natural number value (i.e. drawn from the set $\mathbb{N}$) in each system state.\textsuperscript{4} The states which comprise a schema are called *bindings*, each binding belonging to a schema is a legitimate state of the system. In this example the bindings associate values (of the correct type) to the observations named *volume* and *temp*. We use the word “observation” and never call them “variables”. If one pursues the “schemas as sets of bindings” interpretation (which has been quite standard)

\textsuperscript{4} Note that the schema describes a *state space*, that is, a set of legitimate system states. This is worth stressing because some informal accounts give a mixed message, sometimes suggesting that a schema describes a *particular* state.
then these are constants, not variables. Most informal accounts run into immediate
difficulty in this area.\footnote{See for example [33]. In chapter 11, page 149, they are “variables”; by page 154 they are “components” (constants).}

We will write bindings like this:\footnote{ISO Z uses == rather than \( \Rightarrow \), a notation which dates back to [28] and [29].}

\[
\langle | \text{volume} \Rightarrow n, \text{temp} \Rightarrow m | \rangle
\]
where, in this case, \( n \in \mathbb{N} \) etc. Naturally, it should follow that, for example:

\[
\langle | \text{volume} \Rightarrow 100, \text{temp} \Rightarrow 20 | \rangle \in \text{Jug}
\]
and also:

\[
\langle | \text{volume} \Rightarrow 100, \text{temp} \Rightarrow 200 | \rangle \notin \text{Jug}.
\]

It is possible to extract the values associated with observations from bindings. This
is called binding selection. For example, we should be able to show:

\[
\langle | \text{volume} \Rightarrow 100, \text{temp} \Rightarrow 20 | \rangle. \text{volume} = 100.
\]

In order to capture these ideas we begin by introducing the idea of a schema type:

\[
[\cdots z^T_i \cdots].
\]

This is an unordered sequence of typed (indicated by superscripts) observations
(the \( z_i \)). Then schemas are either schema sets:

\[
[\cdots z_i : C^P_i \cdots]
\]
or they are atomic schemas:

\[
[S \mid P]
\]
where the \( C_i \) are sets, \( S \) is a schema and \( P \) is a predicate.

Of particular note are the carrier sets of the various types. These are formed
by closing:

\[
\mathbb{N} = \{ z^N \mid \text{true} \}
\]
under the cartesian product, power type and schema type operations.\footnote{In fact \( \mathbb{N} \) is only one possible base type. See Section 3 for further details.}

No ambiguity results from the overloading of the symbol \( \mathbb{N} \) here: types appear
only as superscripts – all other uses denote the carrier set.

We have remarked that schemas are sets of bindings. So the logic of schemas
can be obtained from the logic of sets and bindings. In \( Z_C \), for sets, we have:

\[
\frac{P[z/t]}{t \in z \mid P} (\{ \}^+) \quad \frac{t \in z \mid P}{P[z/t]} (\{ \}^-).
\]
Note that $\mathbb{Z}_C$ is strongly typed, so these (typed) set comprehensions present no technical difficulties. See Section 3 for further details.

For bindings, $\mathbb{Z}_C$ has:

$$\emptyset \cdots z_i \in \mathcal{C}_i \cdots \Rightarrow t_i \in t_i \Rightarrow t_i \in \mathcal{C}_i \cdots \Rightarrow t_i \in t_i$$

The first of these establishes what information may be extracted from bindings; the second confirms that these values are all that the binding contains.

The logical rules for schemas flow from the following $\mathbb{Z}_C$ definitions:

$$[\cdots z_i : \mathcal{C}_i \cdots] \equiv \{x | \cdots \land x, z_i \in \mathcal{C}_i \land \cdots\}$$

and

$$[S | P] \equiv \{z \in S | z.P\}.$$

The binding selection operator, introduced in the object logic for selection from bindings (that is, $\mathbb{Z}_C$ terms such as $z.x$) is generalised into a meta-language substitution over terms (that is, meta-terms such as $z.t$) and over propositions (meta-terms such as $z.P$). This is essentially a straightforward structural recursive generalisation of binding selection, and appears in more detail in Section 3 below.

The rules for schema sets are then derivable in $\mathbb{Z}_C$:

$$\cdots t_i \in \mathcal{C}_i \cdots \Rightarrow \{\cdots z_i \in \mathcal{C}_i \cdots\} (\{\}^+) \quad t \in [\cdots z_i : \mathcal{C}_i \cdots] (\{\}^-)$$

and, for atomic schemas:

$$t \in S \quad t.P \quad (S^+)$$

Then for example, writing $b$ for $\{\text{volume} \Rightarrow 100, \text{temp} \Rightarrow 20\}$, we have:

$$100 \in \mathbb{N} \land 20 \in \mathbb{N} \quad b \in [\text{volume} : \mathbb{N}, \text{temp} : \mathbb{N}] (\{\}^+) \quad 100 \leq 250 \land 20 \leq 100 \quad b \in \text{Jug} (S^+)$$

as expected, with the trivial steps omitted.

The elimination rules allow us to determine properties of specifications. For example, taking the product of the temperature and the volume as a rudimentary

\footnote{This is modelled to some extent on the more complex object language substitution \textit{frogspawn} operator to be found in the faulty logic presented in [26]. A thorough analysis of frogspawn terms is presented in [19].}
measure of the thermal energy of the water, we can show that this is never bigger than 25000:
\[
\begin{align*}
\forall b \in \text{Jug} \land b.\text{volume} \leq 250 & \land b.\text{temp} \leq 100 \\
\frac{b.\text{volume} \ast b.\text{temp} \leq 25000}{b \in \text{Jug}}.
\end{align*}
\]

2.2 Schema Algebra and Filtered Bindings

Having now considered simple schemas, we will move on immediately to consider an operation from the schema calculus: schema conjunction.

**Example 2.** Consider the schema expression:

\[
\text{Jug} \land \text{Jug}'.
\]

This is also often referred to as \(\Delta\text{Jug}\) and will be necessary when we consider operation schemas. A precise logical explanation of priming schemas is given below. For now, it is safe to rely on one’s informal understanding.

In order to provide a logical account of schema conjunction, we need to introduce a concept crucial to \(\mathcal{Z}_c\): the type restriction (or filtering) of a binding. Roughly, the bindings we expect in the schema \(S_0 \land S_1\) are those common to \(S_0\) and \(S_1\). But the story is more complicated: the types of \(S_0\) and \(S_1\) (say \(T_0\) and \(T_1\)) need not necessarily be the same. In order for \(S_0 \land S_1\) to be well-defined, these types must agree on their overlap. We will write \(T_0 \supseteq T_1\) (in the meta-theory) for the compatible type union (it is not defined if they are incompatible) of \(T_0\) and \(T_1\). Then, more precisely, the bindings in \(S_0 \land S_1\) will be all the bindings \(z\) in \(T_0 \supseteq T_1\) so that \(z\) restricted to \(T_0\) is a member of \(S_0\), and restricted to \(T_1\) is a member of \(S_1\). Note that when the types are disjoint, this is effectively a union operation.

We write \(z \mid T\) for the \(\mathcal{Z}_c\) term called the restriction (or filtering) of the binding \(z\) to the type \(T\). Naturally it is only well-formed when the type of \(z\) is an extension of \(T\). For example, in \(\mathcal{Z}_c\) we can prove:

\[
\langle x \mapsto 3, y \mapsto 4 \mid \lfloor x^3 \rfloor \rangle = \langle x \mapsto 3 \rangle.
\]

We will write \(T_0 \preceq T_1\) in the meta-theory when \(T_0\) is a schema subtype of \(T_1\) in this sense. The critical \(\mathcal{Z}_c\) rule which effects restricted bindings is this:

\[
\frac{\Gamma, z_i = t_i}{\Gamma | T_1} z_i = t_i} \quad \text{if } T_1 \preceq T_0 \text{ and } z \in \alpha T_1.
\]

The meta-term \(\alpha T\) refers to the (meta-)set of observations occurring in \(T\) (the alphabet of \(T\), see Section 3 below).

A natural generalisation of membership is useful, when \(T_1 \preceq T_0\):

\[
z_{T_0} \in S^{\preceq T_1} = \{z \mid T_1 \in S\}.
\]
This idea can also be applied to equality:

\[ t_0^{T_0} = t_1^{T_1} = df \, t_0 \upharpoonright (T_0 \land T_1) = t_1 \upharpoonright (T_0 \land T_1). \]

Here we have written \( T_0 \land T_1 \) for schema type intersection. The notation is most usefully employed when \( T_1 \subseteq T_0 \) or \( T_0 \subseteq T_1 \).

More generally we have:

\[ t_0^{T_0} = t_1^{T_1} = df \, t_0 \upharpoonright T = t_1 \upharpoonright T. \]

This notation is most usefully employed when \( T \subseteq T_0 \) and \( T \subseteq T_1 \).

With all this in place, we can define schema conjunction by translating the informal description above into a \( Z \mathcal{C} \) definition:

\[ S_0^{P_{T_0}} \land S_1^{P_{T_1}} = df \{ z(P_{T_0 \mathcal{Y} T_1}) : z \upharpoonright T_0 \subseteq S_0 \land z \upharpoonright T_1 \subseteq S_1 \} \]

which leads immediately to the following rules:

\[
\begin{align*}
\frac{t \in S_0 \quad t \in S_1}{t \in S_0 \land S_1} (S_0^+) \\
\frac{t \in S_0 \land S_1}{t \in S_0} (S_0^-) \\
\frac{t \in S_0 \land S_1}{t \in S_1} (S_1^-)
\end{align*}
\]

**Example 3.** Now let us move on to consider operations which change the state. Adding water to the jug:

\[
\begin{array}{l}
\begin{array}{l}
\Delta \text{Jug} \\
more? : [v : \mathbb{N}, t : \mathbb{N}]
\end{array} \\
volume' = volume + more?.v \\
temp' = (volume * temp + more?.v * more?.t) \text{ div } volume'
\end{array}
\]

The declaration in this case amounts to the schema:

\[ \text{Jug} \land \text{Jug'} \land [more? : [v : \mathbb{N}, t : \mathbb{N}]]. \]

Given this observation, no modification of the interpretation of our definition for atomic state schemas is necessary. For example, using the rules already provided (together with other unexceptional rules of equality and propositions) we can prove:

\[ b \in AddWater \]

where \( b \) is the binding:

\[
\downarrow \text{volume} \Rightarrow 50, \text{temp} \Rightarrow 25, more? \Rightarrow m, volume' \Rightarrow 150, temp' \Rightarrow 41 \uparrow
\]

and \( m \) is the binding:

\[
\downarrow v \Rightarrow 100, t \Rightarrow 50 \uparrow.
\]
We have:

\[
\begin{array}{c}
b \in \text{Jug} \\
b \in \text{Jug} \\
\delta \\
\vdots \\
\end{array}
\Rightarrow
\begin{array}{c}
\delta \\
\vdots \\
\end{array}
\]

\[
\begin{array}{c}
b \in \text{Jug} \\
b \in \text{Jug}' \\
\delta \\
\vdots \\
\end{array}
\]

\[
\begin{array}{c}
\delta \\
\vdots \\
\end{array}
\] 

writing \( P \) for \( 150 = 50 + 100 \land 41 = (50 \times 25 + 100 \times 50) \div 150 \) and where, for example, \( \delta \) is:

\[
\begin{array}{c}
100 \in \mathbb{N} \\
150 \in \mathbb{N} \\
\vdots \\
\end{array}
\Rightarrow
\begin{array}{c}
\delta \\
\vdots \\
\end{array}
\]

\[
\begin{array}{c}
m \in \{v : \mathbb{N}, t : \mathbb{N}\} \\
\vdots \\
\end{array}
\]

Example 4. This operation simply takes the temperature of the water in the jug:

\[
\begin{array}{c}
\text{TakeTemp} \\
\Xi \text{Jug} \\
\text{read!} : \mathbb{N} \\
\text{read!} = \text{temp}
\end{array}
\]

This is, as is well-known, shorthand for:

\[
\begin{array}{c}
\text{TakeTemp} \\
\Delta \text{Jug} \\
\text{read!} : \mathbb{N} \\
\text{read!} = \text{temp} \\
\theta \text{Jug} = \theta \text{Jug}'
\end{array}
\]

According to the definition given above, this is interpreted as the following set of bindings in \( \mathcal{Z}_c \):

\[
\{z \in \Delta \text{Jug} \land \text{[more? : [v : N, t : N] \land [read! : N]]} | z.(\text{read!} = \text{temp} \land \theta \text{Jug} = \theta \text{Jug}')\}\}
\]

What is so far missing from our account is an explanation of \( \theta \) -terms. In the unprimed case:

\[
\theta S^{\mathcal{Z}_c - z_i \cdots} S^{\mathcal{Z}_c - z_i - 1} =_{df} \{ \cdots z_i \Rightarrow z_i' \cdots \}.
\]

Thus \( z_0, \theta S^{\mathcal{T}_0} T_1 = z \upharpoonright T_1 \) whenever \( T_1 \preceq T_0 \).

In the primed case we have \( \theta S' = \theta' S \) where:

\[
\theta' S^{\mathcal{Z}_c - z_i \cdots} =_{df} \{ \cdots z_i \Rightarrow z_i' \cdots \}.
\]
The second of these suggests, correctly, that in fact we have an operation (called \( \theta' \)) on \( S \) rather than \( S' \). Indeed, we have not provided a precise explanation of the priming of schemas: \( \theta' \) is the more fundamental concept:

\[
\left[ \cdots x_i : T_i \cdots \right]' =_{df} \left[ \cdots x_i' : T_i \cdots \right]
\]

and:

\[
[S \mid P]' =_{df} [S' \mid \theta' S. P]
\]

The special Z term \( \theta \) has a history of notoriously poor and incomplete explanation. The introduction of characteristic bindings in [33] was a step forward. Integrating this with a comprehensive logic, adding a proper analysis of terms such as \( \theta S' \), in particular the role of the rule \( \Rightarrow \) (see above), provides a complete description of its function and circumstances in which it is properly typed.

2.3 Schema Algebra and Promotion

Promotion is a Z idiom which seeks to bring uniformity (and so security and likelihood of correctness) to a common situation when building models of systems. A similar idea is found with mapping (and its generalisations) as we find in functional programming languages.\(^9\)

In addition to schema conjunction, schema existential quantification (hiding) also makes an appearance in promotion.

Further details of existential quantification appear in Section 3 below. For now, we note that this idea can be formalised in \( Z \) and that the rules for reasoning about such schema expressions are:

\[
\begin{align*}
  t \in S & \quad (S'_{\exists}) \\
  t \in \exists z \in T \bullet S & \\
  t \in \exists z \in T \bullet S : y \in S, y \equiv t \vdash P &
\end{align*}
\]

Let us illustrate promotion by examining the simplest of examples.

Example 5. Consider the following trivial operation:

\[
\text{Inc} \quad v, v' : \mathbb{N} \\
\begin{array}{c}
  v' = v + 1
\end{array}
\]

We wish to promote this operation, over the local state \( \mathbb{N} \), to an operation over the global state \( \mathbb{N} \times \mathbb{N} \). The global operation simply generalises the local operation by

\(^9\) Once again we assume familiarity with practical Z. Promotion is very well introduced and explored in, for example, [33] and [3].
applying it to the first of the pair. The promotion schema as usual explains how the local and global state spaces are to be connected:

\[
\Phi_{\text{Pair}}
\]

\[
\begin{align*}
\Phi_{\text{Pair}} & : v, v' : \mathbb{N} \\
& \wedge w, w' : \mathbb{N} \times \mathbb{N} \\
& w.1 = v \\
& w'.1 = v' \\
& w'.2 = w.2
\end{align*}
\]

and the global operation is:

\[
\text{PairInc} \equiv \exists v, v' : \mathbb{N} \bullet \text{Inc} \wedge \Phi_{\text{Pair}}.
\]

We should, for example, be able to prove that:

\[
\langle \_ \rangle \triangleright(3, 5), \_ \triangleright(4, 5) \rangle \in \text{PairInc}.
\]

We will write this binding as \(b_0\) and the extended binding

\[
\langle \_ \rangle \triangleright 3, \_ \triangleright 4, \_ \triangleright(3, 5), \_ \triangleright(4, 5) \rangle
\]

as \(b_1\). This is straightforward:

\[
\begin{array}{c}
\frac{\delta_0}{b_1 \in \text{Inc} \wedge \Phi_{\text{Pair}} (S^+_x)} \\
\frac{\delta_1}{b_1 \in \text{PairInc} (S^+_x)} \\
\end{array}
\]

Let \(b_2\) be \(\langle \_ \rangle \triangleright 3, \_ \triangleright 4 \rangle\), then \(\delta_0\) is

\[
\begin{array}{c}
\frac{3 \in \mathbb{N} \quad 4 \in \mathbb{N}}{b_1 \in \text{Inc}} \\
\frac{b_1 \in \text{Inc} \wedge \Phi_{\text{Pair}}}{b_2 \in \text{Inc}}
\end{array}
\]

and \(\delta_1\) is

\[
\begin{array}{c}
\frac{b_1 \in [v, v' : \mathbb{N}, w, w' : \mathbb{N} \times \mathbb{N}] \quad 3 = 3 \wedge 4 = 4 \wedge 5 = 5}{b_1 \in \Phi_{\text{Pair}}}
\end{array}
\]

Here we omit all trivial steps, and those previously illustrated. Naturally this proof illustrates the direct use of the basic rules for schema expressions, schemas and the
base logic itself. As with all logics, it is in practice necessary to develop further derived rules to streamline derivation.

One can, of course, also reason from complex expressions (using the elimination rules). The following example shows that the second part of the global state is always unchanged. This trivial example is a prototype for the general policy of determining general properties which complex specifications enjoy:

**Example 6.** Consider the following property

\[ \forall b \in \text{PairInc} \cdot b.w.2 = b.w'.2 \]

and the proof, which uses the elimination rules for existential, conjunctive and atomic schemas is:

\[
\begin{array}{c}
\frac{y \in \text{Inc} \land \Phi\text{Pair}} {y \in \Phi\text{Pair}}^2 \\
\frac{y.w.1 = y.v \land t.w'.1 = y.v' \land y.w.2 = y.w'.2} {y.w.2 = y.w'.2}^1 \\
\frac{b \in \text{PairInc} } {b = b}^1 \\
\frac{b.w.2 = b.w'.2}{b.w.2 = b.w'.2}^2, (S^-) \\
\frac{\forall b \in \text{PairInc} \cdot b.w.2 = b.w'.2}{ \forall b \in \text{PairInc} \cdot b.w.2 = b.w'.2}^1
\end{array}
\]

### 3 THE SPECIFICATION LOGIC \(\mathcal{Z}_c\)

\(\mathcal{Z}_c\) is an extension of higher order logic with the addition of the *schema types* we introduced above.

#### 3.1 The Types of \(\mathcal{Z}_c\)

We begin with the language of types:

\[ T ::= \Upsilon \mid \mathbb{P} T \mid T \times T \mid \cdots z^T \cdots . \]

Types of the form \(\Upsilon\) are the names of *free types* and are given by equations of the form

\[ \Upsilon ::= \cdots \mid c_i \langle \cdots \Upsilon_y \cdots \rangle \mid \cdots \]

where any of the \(\Upsilon_y\) may be \(\Upsilon\) (permitting recursion). In particular, \(\langle \cdots \Upsilon_y \cdots \rangle\) may be omitted. An important example is

\[ N ::= \text{zero} \mid \text{succ} \langle \text{N} \rangle. \]
This class of free types is quite simple, but has the virtues of covering many practical cases and ensuring the existence of trivial set theoretic models. We do not permit mutual recursion here, but the generalisation is straightforward.\footnote{For the reader interested in pursuing the technical issues concerning free-types, see [2, 30] for example.}

Types of the form $[\cdots z_T^i \cdots]$ (the order is not important) are called \textit{schema types}. We write $\alpha[\cdots z_T^i \cdots]$ for the alphabet set (in the meta-language) of observations $\{\cdots z, \cdots\}$. No observation may occur more than once in such a type. The symbols $\preceq$, $\prec$, $\succ$ and $-$ denote the \textit{schema subtype} relation, and the operations of \textit{schema type intersection}, \textit{schema type union} and \textit{schema type subtraction}. All these relations and operations are defined only for schema types, so any future definition which makes use of them is only well-defined when the types in question are schema types. Schema type union imposes an additional constraint, since it is only defined when its schema type arguments are \textit{compatible} (common observations agree on their type).

The last important operation on types is \textit{priming}. First we associate with every observation $z$ its \textit{co-observation} $z'$ where $z'' = z$. Then we set $[\cdots z \cdots]'$ to be $[\cdots z' \cdots]$. This is not a convention of vernacular $Z$ but turns out to be extremely useful in $Z$ logic, especially when combined with pattern matching syntax in definitions.\footnote{Much use of the idea of treating priming to be an operation, rather than a diacritical, is made in Section 4.8 (the definition of composition) and in Section 5.3, especially in connection with data refinement and the definitions of simulation images and co-images.}

All further syntactic categories of the language of $Z_C$ must be well-formed with respect to these types. Types are indicated by superscripting and omitted whenever possible.

We now move on to describe the languages of terms and propositions and their corresponding logical rules. The judgements of $Z_C$ have the form $\Gamma \vdash P$ where $\Gamma$ is a set of formulae. The logic is presented as a natural deduction system \textit{in sequent form}. We shall omit all data (entailment symbol, contexts, type etc.) which remain unchanged by any rule.

\section*{3.2 The Terms of $Z_C$}

First we have variables, bindings, pairs and their projections:\footnote{The reader may already have noticed, from examples in Section 2, that we carefully distinguish \textit{observation meta-variables} and \textit{variable meta-variables}. In the \textit{object language} we do not make any distinction. The latter is quite standard in vernacular $Z$ and the former ensures that the potential ambiguity is resolved at the level of the syntax.}

\begin{align*}
t^T & ::= x^T \mid t^{[\cdots z \cdots]}, z \mid t^{T \times T_1}, 1 \mid t^{T_0 \times T}, 2 \\
t^{T_0 \times T_1} & ::= (t^{T_0}, t^{T_1}) \\
t^{[\cdots z \cdots]} & ::= \{ \cdots z \Rightarrow t \cdots \}
\end{align*}
These terms are characterised by various logical rules:

\[
\frac{\langle \cdots z_i \rangle \Rightarrow t_i \cdots}{\langle \cdots z_i \rangle \Rightarrow t_i \cdots} (\Rightarrow^+) \\
\frac{\langle \cdots z_i \rangle \Rightarrow t_i \cdots}{t^{1-z_i^{1-\cdot}}} (\Rightarrow^-)
\]

Second, we have the filtered (restricted) bindings.

\[
t^T_0 := t^T_0 \upharpoonright T_0 \quad \text{where } T_0 \subseteq T_1
\]

As we have seen, the rule for these is:

\[
\frac{t^T_0, z_i = t_i (\downarrow^+)}{t^T_1, z_i = t_i \ (\downarrow^2)} T_1 \subseteq T_0 \text{ and } z \in \alpha T_1.
\]

Third, there are the values of free-type:

\[
t^\Upsilon := c_i \cdots t^\Upsilon_0 \cdots
\]

The logic of free types permits the introduction of values in the type, equality reasoning and, finally, elimination (generally by induction).

\[
\begin{align*}
\cdots z_j \in \Upsilon_0 \cdots &\quad c_i \cdots z_j \cdots \in \Upsilon (\Upsilon^+) \\
\cdots z_j \in \Upsilon_0 \cdots &\quad \cdots z_k \in \Upsilon_1 \cdots (\Upsilon^-)
\end{align*}
\]

where the \( y_k \) are all those variables occurring in the \( z_j \) with type \( \Upsilon \).

Finally, we have sets:

\[
t^{PT} := \{ z^T \ | \ P \}.
\]

These are governed by:

\[
\frac{P[z/t]}{t \in \{ z \ | \ P \}} (\{ \}^+) \\
\frac{t \in \{ z \ | \ P \}}{P[z/t]} (\{ \}^-)
\]

For clarity of presentation we will use the meta-variable \( C \) (etc.) for sets (terms of power type), and \( S \) (etc.) for sets of schema type. The latter are, as we have seen, the schemas.
M. C. Henson, S. Reeves, J. P. Bowen

We employ the notation $b. P$ and $b. t$ (generalising binding selection) which is adapted from [32]. Suppose that \{\ldots z_i \ldots\} is the alphabet set of $t$, then the following equation holds:

$$ t. P = P[\ldots z_i \ldots / \ldots t. z_i \ldots] $$

\section*{3.3 The Formulae of $Z_C$}

The formulæ of $Z_C$ delineate a typed bounded predicate logic.

$$ P ::= false \mid t^T = t^T \mid t^T \in C^P \mid \neg P \mid P \lor P \mid \exists z^T \in C^P \cdot P $$

The logic of $Z_C$ is classical, so the remaining logical operations are available by definition. We also, as usual, abbreviate $\neg(t \in C)$ to $t \not\in C$.

A crucial observation is \textit{uniquity of types}: every term of $Z_C$ has a unique type. We can make great use of this observation. It enables us to remove type decoration in most circumstances.

The logic for the propositions is then standard:

$$ \frac{P_0 \lor P_1}{P_0 \lor P_1} (\lor^+) $$

$$ \frac{P_0 \lor P_1 \mid P_1 \vdash P_2}{P_0 \lor P_1 \mid P_2 \vdash P_2} (\lor^-) $$

$$ \frac{P \vdash false}{\neg P} (\neg^+) $$

$$ \frac{P \vdash false}{\neg P} (false^+) $$

$$ \frac{\exists z \in C \cdot P_0 \mid y \in C, P_0[z/y] \vdash P_1}{P_1} (\exists^-). $$

The eigenvariable $y$ may not, as usual, occur in $C, P_0, P_1$ nor any other assumption.

$$ \frac{\Gamma, P \vdash P}{\Gamma, \top = \top} (ass) $$

$$ \frac{t_0 = t_1 \mid P[z/t_0]}{P[z/t_1]} (sub) $$

$$ \frac{t_0 \equiv t_1 \mid t_0 = t_1}{t_0 \equiv t_1} (ext) $$

where:

$$ t_0 \equiv t_1 =_{df} \forall z \in t_0 \cdot z \in t_1 \land \forall z \in t_1 \cdot z \in t_0. $$

The transitivity of equality and numerous \textit{equality congruence} rules for the various term forming operations are all derivable in view of rule \textit{(sub)}. In particular, we can prove that set-equality in $Z_C$ is extensional.

As an example of the rules for free types we can give the following specialisations for $N$, as defined above:

$$ zero \in N $$

$$ n \in N $$

$$ n \in N $$

$$ zero \neq succ \ n $$
\[
\text{succ } n = \text{succ } m \quad \frac{\text{n } = \text{m}}{P[n/\text{zero}] \quad m \in \mathbb{N}, \quad P[n/m] \vdash P[n/\text{succ } m]} \quad n \in \mathbb{N} \vdash P
\]

The following weakening rule is admissible and is incorporated within the system:

\[
\frac{\Gamma \vdash P_1}{\Gamma, P_0 \vdash P_1} \quad (wk)
\]

Finally, a term of type \( T \) always belongs to the carrier set of \( T \):

\[
\overline{t}^T \in T.
\]

### 3.4 Consistency

The only interesting issue is the interpretation of schema types and bindings, including binding selection and filtering.

Let \( B \) be an \( I \)-indexed family of sets over a suitable universe \( U \). We can define a dependent function space which is suitable for our purposes as follows:

\[
\Pi_{(i \in I)}.B(i) = \{ f \in I \rightarrow U \mid (\forall i \in I)(f(i) \in B(i)) \}. 
\]

This we can harness to interpret the schema types of \( Z \):

\[
\llbracket \cdots z_i \cdots \rrbracket =_{df} \Pi_{x \in I}.B(x)
\]

where \( I =_{df} \{ \cdots z_i \cdots \} \) and \( B(z_i) =_{df} \llbracket T_i \rrbracket \). The observations \( z_i \) can be modelled in ZF in any number of ways, for example as finite ordinals. The only important point is that they be distinguishable from one another.

Then bindings, binding projection and filtered terms are defined as follows:

\[
\begin{align*}
\llbracket \cdots z_i \leftrightarrow t_i \cdots \rrbracket & =_{df} f_0 \\
\llbracket t, z \rrbracket & =_{df} \llbracket t \rrbracket (z) \\
\llbracket t \mid T \rrbracket & =_{df} f_1
\end{align*}
\]

where \( f_0 \in \llbracket \cdots z_i \cdots \rrbracket \), \( f_0(z_i) = \llbracket t_i \rrbracket \), \( f_1 \in \llbracket T \rrbracket \) and \( f_1(z) = \llbracket t \rrbracket (z) \) when \( z \in \alpha[D] \).

Further detail is provided in [17] and (for free-types) in [19].

### 3.5 An Alternative Approach

The system we have described is a “Church-style” theory, in which the syntax formation rules are controlled by typing considerations and where terms explicitly carry their types. The unicity of types does simplify matters, permitting types to be omitted in most circumstances. The meta-language is imposed upon to carry the burden
of this. Naturally a machine implementation of the logic would need to consider these issues explicitly.

An alternative “Curry-style” approach was described in [17] and [18]. In that presentation neither terms nor propositions were type controlled. The logic, in that context, comprises two linked theories of typing and inference. This has the effect of making the logic as a whole considerably more complex, though the added explicit information might well be more convenient as a basis for a machine implementation.

In the “Curry-style” system one has an additional judgements of the form $\Gamma \vdash P \ \text{prop}$ and $\Gamma \vdash t : T$. There are then typing rules such as:

$$
\frac{t_0 : T \quad t_1 : T}{t_0 = t_1 \ \text{prop}} \quad (C_\omega)
\frac{t : T \quad C : \mathbb{P} T}{t \in C \ \text{prop}} \quad (C_\varepsilon)
$$

These rules ensure that well-formed equality statements are between terms of the same type and that well-formed membership propositions are also appropriately typed.

We also have rules for non-atomic propositions such as:

$$
\frac{P_0 \ \text{prop} \quad P_1 \ \text{prop}}{P_0 \lor P_1 \ \text{prop}} \quad (C_V)
\frac{x : T \ \text{prop} \quad \exists x : T \bullet P \ \text{prop}}{\exists x : T \bullet P \ \text{prop}} \quad (C_3).
$$

With these in place the logical rules can be stated. These typically make reference to typing judgements. For example:

$$
\frac{\Gamma \vdash P_0 \quad \Gamma^- \vdash P_1 \ \text{prop}}{\Gamma \vdash P_0 \lor P_1 \ \text{prop}} \quad (\lor^+_{a})
$$

and:

$$
\frac{\Gamma \vdash P[z/t] \quad \Gamma^- \vdash t : T}{\Gamma \vdash \exists z : T \bullet P \ \text{prop}} \quad (\exists^+).
$$

In these rules, the context $\Gamma^-$ represents the restriction of the context $\Gamma$ to its typing assertions only.

In this version of the logic one has the following critical result concerning syntactic consistency:

If $\Gamma \vdash P$ then $\Gamma^- \vdash P \ \text{prop}$.

This is proved by induction on the structure of the derivation $\Gamma \vdash P$.

### 4 CONSERVATIVE EXTENSIONS

The base logic $\mathcal{Z}_C$ contains only rudimentary features of $\mathcal{Z}$ (schema types and bindings). We have, in Section 2, indicated in overview how $\mathcal{Z}_C$ can host more advanced features by means of conservative extensions. This approach is simple and attractive, in particular the question of the consistency of more complex features is automatic.
4.1 Schema Sets and Atomic Schemas

Let \( T = [\cdots, z_i^T, \cdots] \). The syntax of basic schemas is:

\[
S^T \ ::= \ [\cdots, z_i^C, T_i^T, \cdots] \mid [S^T | P].
\]

These are the schema sets and atomic schemas, respectively. As usual, we will write schemas of the form:

\[
[S | P].
\]

We allow the obvious generalisation of our alphabet operator to atomic state schemas and state schema sets:

\[
\alpha[S | P] = df \alpha S \text{ and } \alpha[\cdots, z_i^C, T_i^T, \cdots] = df \alpha[\cdots, z_i^T, \cdots].
\]

Then these two basic schemas can be interpreted in \( \mathcal{Z}_C \) as follows:

14

\[
[S | P] = df \{z \in S | z . P\}.
\]

As we have already seen, the rules for schema sets are:

\[
\frac{\cdots \ t_i \in C_i \ \cdots}{\cdots \ z_i \in [\cdots, z_i^C, T_i^T, \cdots]} \text{ (}[[\_]]^+) \quad \frac{t \in [\cdots, z_i^C, T_i^T, \cdots]}{t, z_i \in C_i} \text{ (}[[\_]]^-)
\]

and, for atomic schemas:

\[
\frac{t \in S}{t \in [S | P]} \text{ (}S^+) \quad \frac{t \in S}{t \in [S | P]} \text{ (}S^-) \quad \frac{t, P}{t \in [S | P]} \text{ (}S^-).
\]

There is an important point to make regarding the interpretation of schemas: the proposition \( P \) appearing in a schema is drawn from a more permissive grammar of propositions than that established for \( \mathcal{Z}_C \). In particular, propositions in that context can contain observations as terms. A simple example will suffice to illustrate this.

Example 7. Consider the following schema:

\[
\begin{array}{c}
Inc \\
v, v' : \mathbb{N} \\
v' = v + 1
\end{array}
\]

Consultation of the syntax of \( \mathcal{Z}_C \) will reveal that the proposition \( v' = v + 1 \) is not a \( \mathcal{Z}_C \) proposition because the observations \( v \) and \( v' \) are not terms of \( \mathcal{Z}_C \). This generality

14 Strictly speaking we should indicate (both here and below) the translation explicitly, writing for example:

\[
[[S | P]] = df \{z \in [S] | z . P\}
\]

We will not bother with this as the intention is always quite obvious, and the use of the extra brackets is notationally very burdensome.
in the specification language is perfectly acceptable in view of the interpretation of schemas. Pursuing this example, the $\mathcal{Z}_C$ interpretation is:

\[ \{ z^{|v_{\{\alpha_{\mathcal{E}}}]} \mid z.(v' = v + 1) \} \]

which simplifies to:

\[ \{ z^{|v_{\{\alpha_{\mathcal{E}}}]} \mid z.v' = z.v + 1 \} \]

Note that $z.v' = z.v + 1$ is a 

\textit{bona fide} proposition in $\mathcal{Z}_C$. In all cases a schema proposition $P$ becomes $z.P$ under the interpretation and $z.P$ will always be well-defined.

\subsection*{4.2 $\theta$-terms}

The special Z term $\theta$ is interpreted as described in Section 2.2:

\[ \theta S^{|z_{\{1\} \cdots |}} = df \langle | \cdots \ z_i \Rightarrow z_i \cdots | \rangle \]

In the \textit{primed} case we have $\theta S' = \theta S'$ where:

\[ \theta' S^{|z_{\{1\} \cdots |}} = df \langle | \cdots \ z_i \Rightarrow z_i' \cdots | \rangle. \]

It is also worth noting that these special terms are not in themselves $\mathcal{Z}_C$ terms, but will translate under the interpretation appropriately. Another example:

\textbf{Example 8.} Consider the following schemas:

\begin{center}
\begin{tabular}{c c c}
\hline
\textit{Example} & $\Delta S$ & $\theta S = \theta S'$ \\
\hline
\end{tabular}
\end{center}

where:

\begin{center}
\begin{tabular}{c c c}
\hline
$S$ & $v : \mathbb{N}$ \\
\hline
\end{tabular}
\end{center}

Under the interpretation we will have:

\[ \{ z^{|v_{\{\alpha_{\mathcal{E}}}]} \mid z.(\theta S = \theta S') \}, \]

and this will simplify to:

\[ \{ z^{|v_{\{\alpha_{\mathcal{E}}}]} \mid z.v = z.v' \} \]

This is as expected, and the proposition $z.v = z.v'$ contains well-formed $\mathcal{Z}_C$ terms.
4.3 Schema Disjunction

When the schemas $S_0$ and $S_1$ have the types $\mathbb{P}^T_0$ and $\mathbb{P}^T_1$, the schema expression $S_0 \lor S_1$ has the type $\mathbb{P}(T_0 \lor T_1)$. The definition of schema disjunction in $\mathbb{ZC}$ is:

$$S_0^p \lor S_1^p = \{ z^{p(T_0 \lor T_1)} \mid z \in S_0 \lor z \in S_1 \}.$$ 

This leads to the following rules:

$$\frac{t \in S_0}{t \in S_0 \lor S_1} (S_0^+) \quad \frac{t \in S_1}{t \in S_0 \lor S_1} (S_1^+).$$

4.4 Schema Conjunction

When the schemas $S_0$ and $S_1$ have the types $\mathbb{P}^T_0$ and $\mathbb{P}^T_1$, the schema expression $S_0 \land S_1$ has the type $\mathbb{P}(T_0 \land T_1)$. The definition of schema conjunction in $\mathbb{ZC}$ is, as we have seen:

$$S_0^p \land S_1^p = \{ z^{p(T_0 \land T_1)} \mid z \in S_0 \land z \in S_1 \},$$

and the rules are:

$$\frac{t \in S_0 \quad t \in S_1}{t \in S_0 \land S_1} (S_0^+) \quad \frac{t \in S_0 \land S_1}{t \in S_0} (S_0^-) \quad \frac{t \in S_0 \land S_1}{t \in S_1} (S_1^-) \quad \frac{t \in S_0 \land S_1}{t \in S_0 \land S_1} (S_0^-).$$

4.5 Schema Negation

Schema negation is straightforward:

$$\neg S^p = \{ z^T \mid z \notin S \},$$

and these rules follow:

$$\frac{t \notin S}{t \in \neg S} (S^+) \quad \frac{t \in \neg S}{t \notin S} (S^-).$$

4.6 Schema Inclusion

In addition, our notion of atomic schemas combines with schema conjunction to provide an immediate treatment of schema inclusion by interpreting the separation of declarations in a schema as schema conjunction. For example, the schema $[z : T ; S \mid P]$ is just $[z : T \land S \mid P]$ and so on.
4.7 Schema Existential Hiding

If the schema $S$ has the type $P \cdot T_1$ and $[z^{T_0}] \preceq T_1$, then the type of the schema expression $\exists z : T_0 \cdot S$ is $P(T_1 - [z^{T_0}])$.

Existentially quantified schemas are interpreted in $Z_C$ as follows:

\[ \exists z : T_0 \cdot S =_{df} \{ x \in T_1 - [z^{T_0}] \mid \exists y \in T_1 \cdot y \in S \land x = y \upharpoonright (T_1 - [z^{T_0}]) \}; \]

then these logical rules follow:

\[ \frac{t \in S}{t \in \exists z : T \cdot S} (S_0^+) \]
\[ \frac{t \in \exists z : T \cdot S \quad y \in S, y \models t \vdash P}{y} (S_0^-) \]

4.8 Schema Composition

In this and the next section we will consider operation schema, that is, those schemas whose type is $P \cdot T$ where $T$ has the form $T_{in} \upharpoonright T_{out}$ where $T_{in}$ contains declarations of all before observations and $T_{out}$ contains declarations of all after observations. We will also need to refer to $T_{out}'$, the co-type of $T_{out}$. We will use the meta-variable $U$ when we specifically refer to operation schemas.

Note that the types $T_{in}$ and $T_{out}'$ are always disjoint. We can therefore write the bindings belonging to $U$ in the form $t_0 \star t_1'$ where $t_0$ has type $T_{in}$, where $t_1'$ has the type $T_{out}'$ and where the star represents binding concatenation which will only be defined in circumstances in which its arguments have non-overlapping type. This operation can be raised to sets:

\[ C_0 \star C_1 =_{df} \{ z_0 \star z_1 \mid z_0 \in C_0 \land z_1 \in C_1 \}. \]

For schema composition we present only a special case. For the general case (which is substantially more complex) and for related operations, like schema piping, see [19]. Suppose $T_{out}' = T_{in}$. Then:

\[ U_0^{T_{in} \upharpoonright T_{out}'} \upharpoonright U_1^{T_{in} \upharpoonright T_{out}'} =_{df} \{ (z_0 \star z_1')^{T_{in} \upharpoonright T_{out}'} \mid \exists y^{T_{out}'} \cdot z_0 \star y' \in U_0 \land y \star z_1' \in U_1 \}. \]

The rules are then:

\[ \frac{t_0 \star t_1' \in U_0 \quad t_2 \star t_1' \in U_1}{t_0 \star t_1' \in U_0 \upharpoonright U_1} (U_0^+) \]
\[ \frac{t_0 \star t_1' \in U_0 \upharpoonright U_1 \quad t_0 \star y' \in U_0, y \star t_1' \in U_1 \vdash P}{P} (U_0^-) . \]

The usual sideconditions apply to the eigenvariable $y$. 
4.9 Schema Precondition

We can introduce the idea of the precondition of an operation schema (essentially the domain of the partial relation the schema denotes).

Let $T^m \preceq V$. Then:

$$\text{Pre } U \ x^V \ =_d \exists z \in U \cdot x =_{T^m} z.$$  

This leads to the following rules:

$$\frac{t_0 \in U \quad t_0 =_{T^m} t_1}{\text{Pre } U \ t_1} \quad (\text{Pre}^+) \quad \frac{\text{Pre } U \ t \quad y \in U \cdot y =_{T^m} t \vdash P}{P} \quad (\text{Pre}^-)$$

where $y$ is fresh.

For later convenience, the notion of precondition is introduced as a predicate. In vernacular Z the precondition is a schema (set of bindings). This is easily recovered when necessary as $\{z^{T^m} \mid \text{Pre } U \ z\}$.

The reader interested in pursuing these issues in further depth, for example for more general operations such as schema level quantification and generic schemas, should see [17, 18, 19] which contain more detail.

5 BEYOND SPECIFICATION

In this section we provide an overview and survey of a number of topics which build still further on Z logic. Once Z has been established as a specification logic it becomes possible to address new issues and characteristic properties in a systematic and integrated manner. We will begin with the most familiar: the equational logic of Z and the precondition logic for schema expressions. After this we tackle the crucial topic of refinement. With all this in place it becomes possible to investigate the monotonicity (or otherwise) of the schema calculus operators with respect to refinement.

Our treatment here is necessarily brief and incomplete: readers who consult the relevant literature will find not only more detail concerning the topics addressed here, but also many other investigations which we have not mentioned here at all. Our purpose in this section is to whet the reader’s appetite through our summary and survey. Only the main contours of the topics addressed are highlighted and readers will need to rely on their general knowledge of the topics discussed.

5.1 Equational Logic

It is interesting to note that the fundamental relation of Z is, in fact, equality. So far as schemas are concerned, this is essentially equality of the partial relations which (in particular, operation) schemas denote.

In the absence of a logic, the informal explanation of schema operators has often been given in terms of certain equalities.
Example 9. It is case that:

\[ [T_0 \mid P_0] \land [T_1 \mid P_1] = [T_0 \land T_1 \mid P_0 \land P_1]. \]

Note that this equality is not definitional. In the context of the logic it should be (and indeed is) derivable.

This, and all other expected schema equations, are derivable in the schema logic described in Section 3. By way of example, consider the expected equation for negated schemas.

\[ \neg [T \mid P] = [T \mid \neg P]. \]

This is the proof: The result then follows, by rule \((ext)\), from these two derivations:

\[
\begin{array}{c}
\frac{t \in \neg [T \mid P]}{t \notin [T \mid P]} (S^-_\neg) \\
\frac{t \notin T}{\neg (t \in T \land t.P)} false \\
\frac{\neg (t \in T \land t.P)}{t \notin T \lor \neg t.P} (S^+_\neg)
\end{array}
\]

and:

\[
\begin{array}{c}
\frac{t \in [T \mid \neg P]}{\neg t. P} (S^-_\neg) \\
\frac{\neg t. P}{t \notin [T \mid P]} false \\
\frac{t \notin [T \mid P]}{t \in \neg [T \mid P]} (S^+_\neg)
\end{array}
\]

5.2 Precondition Logic

We considered the concept of schema precondition in Section 4.9. That general logical account can be combined with the logic of the schema calculus to provide a logic of schema preconditions for all compound schemas.

5.2.1 The Precondition for Conjunction Schemas

In general, the precondition of a conjunction of operations is not the conjunction of the preconditions of the individual constituents [31]. This is a direct consequence of the underlying “postcondition only” approach \(Z\) takes (in contrast to other notations such as B [1] or the refinement calculus [25]).

Let \(i \in \{0, 1\}\), then the following elimination rules are derivable for the precondition of conjoined schemas:

\[
\frac{Pre \ (U_0 \land U_1) t}{Pre \ U_i t} (Pre^*_i).
\]
5.2.2 The Precondition for Disjunction Schemas

The analysis of the precondition of disjoined operations is more straightforward. Let \( i \in \{0, 1\} \), then the following introduction and elimination rules for the precondition of the disjunction of schemas are derivable:

\[
\frac{\text{Pre } U \_i \ t}{\text{Pre } (U_0 \lor U_1) \_i \ t \ (\text{Pre}_i^+)}
\]

\[
\frac{\text{Pre } U_0 \_0 \ t \ P \ \text{Pre } U_1 \_1 \ t \ P}{\text{Pre } (U_0 \lor U_1) \_0 \ t \ (\text{Pre}_0^-) \ \text{Pre } (U_0 \lor U_1) \_1 \ t \ (\text{Pre}_1^-)}
\]

With these in place, we can easily prove the full distributivity of the precondition over disjunction.

\[
\text{Pre } (U_0 \lor U_1) \_i \ t \leftrightarrow \text{Pre } U_0 \_i \ t \lor \text{Pre } U_1 \_i \ t
\]

5.2.3 The Precondition for Composition

We will deal with instances of composition where the operation schema expression \( U_0 \circ U_1 \) has the type \( P(T_0 \bowtie T'_1) \) and where \( U_0 \) is of type \( P(T_0 \bowtie T'_2) \) and \( U_1 \) is of type \( P(T_2 \bowtie T'_1) \).

The following introduction and elimination rules for the precondition of composed operation schemas are derivable:

\[
\frac{l_0 \star l_1 \in U_0 \text{ Pre } U_1 \_i \ t}{\text{Pre } (U_0 \circ U_1) \_0 \ t \ (\text{Pre}_0^+)}
\]

\[
\frac{\text{Pre } (U_0 \circ U_1) \_0 \ t \text{ Pre } U_1 \_i \ y, l_0 \star l_1 \in U_0 \ldash P}{\text{Pre } (U_0 \circ U_1) \_0 \ t \ (\text{Pre}_0^-)}
\]

The usual side-conditions apply to the eigenvariable \( y \).

The following additional rule is derivable for the precondition of composition:

\[
\frac{\text{Pre } (U_0 \circ U_1) \_0 \ t}{\text{Pre } U_0 \_0 \ t}
\]

5.2.4 The Precondition for the Existential Quantifier

In this case we consider the simultaneous hiding of an observation and its co-observation in an operation. Let \( z \) (and \( z' \)) have the type \( T^z \). Then we can derive the following rules:

\[
\frac{\text{Pre } U \_t}{\text{Pre } (\exists z, z' : T^z \cdot U) \_t \ (\text{Pre}_z^+)}
\]

\[
\frac{\text{Pre } (\exists z, z' : T^z \cdot U) \_t \text{ Pre } U \_y, y = t \ldash P}{\text{Pre } U \_y \_t \ (\text{Pre}_z^-)}
\]

Note that the usual side-conditions apply to the eigenvariable \( y \).

Further detail, including a treatment of other schema operations, can be found in [12].
5.3 Refinement Logic

The ordinary subset relation on schemas (sets of bindings) establishes a primitive theory of refinement for Z. It is, however, unacceptable for it is only a partial correctness theory and it treats preconditions as firing conditions. To see this, note first that the empty set of bindings is a subset of all sets of bindings of the appropriate type and therefore a refinement of all such schemas. This schema establishes no conditions whatsoever and well-defined input/output relations will be lost in such a refinement. This is, then, evidently a partial correctness model. Second, note that weakening the precondition can introduce new input/output relationships which were not previously present. Clearly adding new relationships to a set does not lead to a subset, and hence not to a refinement. Evidently this is a theory of refinement for firing conditions.

The standard total correctness theory of refinement (also permitting weakening of preconditions) involves the process of relational completion (see for example [33], Chapter 16 et seq.). This completion is often called the lifted-totalisation and introduces an additional element usually written \( \bot \). Such a value must be separated from the interpretation of Z and this is easily achieved by introducing a simple \( Z_{\bot} \) theory which we call \( Z_{\bot} \).

In \( Z_{\bot} \) we introduce new constants (“abortive” values), postulating new constants \( \bot^T \) for every type \( T \): these are usually called “lifted” types. There are, additionally, a number of axioms which ensure that all the new \( \bot^T \) values interact properly.

\[
\begin{align*}
\{ z_0 \Rightarrow \bot^T_0 \ldots z_n \Rightarrow \bot^T_n \} &= \bot^{[z_0 \ldots z_n]} \\
(\bot^T_0 \times \bot^T_1) &= \bot^T_0 \times T_1, \\
\{ z^T \mid z = \bot^T \} &= \bot^T \times T
\end{align*}
\]

For example:

\[ \bot^{[z_0 \ldots z_n]} \cdot z_i = \bot^T_i, \quad (0 \leq i \leq n). \]

These are the only axioms concerning these terms, hence, the term forming constructions are non-strict with respect to the \( \bot^T \) values.

Natural carriers for each type (sets which exclude \( \bot \)) are then easily defined by closing:

\[ T \equiv \{ z^T \mid z \neq \bot \} \]

under the type forming operations. These are then used to establish the (\( \bot \)-free) schema logic, as described in Section 3 above.

Further details, including the fact that the theory \( Z_{\bot} \) is conservative over \( Z \), can be found in [11].

The lifted totalisation of a set of bindings can then be defined. Let

\[ T_\bot = T \cup \{ \bot \} \]
and

\[ T^* = \text{df} T_{\perp}^{in} \times T_{\perp}^{out}; \]

then:

\[ U = \text{df} \{ z_0 \times z'_1 \in T^* \mid \text{Pre } U z_0 \Rightarrow z_0 \times z'_1 \in U \} \]

which leads to rules for lifted totalised sets:

\[ \frac{t_0 \times t'_1 \in T^* \quad \text{Pre } U t_0 \vdash t_0 \times t'_1 \in U}{t_0 \times t'_1 \in \hat{U}} (\star^+ ) \]

and

\[ \frac{t_0 \times t'_1 \in \hat{U} \quad \text{Pre } U t_0}{t_0 \times t'_1 \in U} (\star^- ) \]

The following are also derivable:

\[ \frac{U \subseteq \hat{U}}{(i) \quad \perp \in U} \quad (\text{ii} \quad \neg\text{Pre } U t_0 \vdash t_0 \in T_{\perp}^{in} \quad t'_1 \in T_{\perp}^{out'}} \]

(\star^- )

These demonstrate that the definition is consistent with the usual intentions: the underlying partial relation is contained in the completion, the entirely abortive binding is present in the relation, and, more generally, each value outside the precondition (including \( \perp \)) maps to every value in the co-domain of the relation.

5.3.1 Operation Refinement

We first consider operation refinement in which the data-types involved do not change.

\( W^- \)-refinement, written \( U_0 \geq_{w^-} U_1 \) is defined by:

\[ U_0 \geq_{w^-} U_1 = \text{df} \quad \hat{U}_0 \subseteq \hat{U}_1 \]

Obvious introduction and elimination rules follow from this.

In fact the rather complex manoeuvres necessary to set up this definition are unnecessary: refinement can be captured entirely in terms of the language of \( Z \) itself. Let \( z, z_0, z_1 \) be fresh variables:

\[ \frac{\text{Pre } U_1 z \vdash \text{Pre } U_0 z \quad \text{Pre } U_1 z_0 \times z'_1 \in U_0 \vdash z_0 \times z'_1 \in U_1}{U_0 \geq_{w} U_1} (\exists^+ ) \]

\[ \frac{U_0 \geq_{w} U_1 \quad \text{Pre } U_1 t \quad \text{Pre } U_0 t}{U_0 \geq_{w} U_1} (\exists^- ) \]

\[ \frac{U_0 \geq_{w} U_1 \quad \text{Pre } U_1 t_0 \times t'_1 \in U_0}{t_0 \times t'_1 \in U_1} (\exists^- ) \]
The theories $\sqsupseteq_w$ and $\sqsupseteq_s$ are equivalent (they are the same relation on specifications) [11].

Other refinement approaches for $Z$, such as a weakest preconditions (wp) approach, can also be formalised in $\mathcal{Z}_C$.

First we have post-sets:

$$\text{Post } U \ z_0 = \text{df } \{ z'_1 \mid z_0 \star z'_1 \in U \}$$

This permits a wp-interpretation for schemas

$$\text{wp } U \ C = \text{df } \{ z \mid \text{Pre } U \ z \land \text{Post } U \ z \subseteq C \}$$

leading to the following rules:

$$\begin{array}{c}
\text{Pre } U \ t \ z' \in \text{Post } U \ t \vdash z' \in C \\
t \in \text{wp } U \ C
\end{array}$$

where $z$ is a fresh variable.

$$\begin{array}{c}
t \in \text{wp } U \ C \\
 t_0 \in \text{wp } U \ C \\
 t'_1 \in \text{Post } U \ t_0 \\
 t'_1 \in C
\end{array}$$

We can now define WP-refinement:

$$U_0 \sqsupseteq_{wp} U_1 = \text{df } \forall \ C^p \ T^\bot \bullet \ \text{wp } U_1 \ C \subseteq \text{wp } U_0 \ C$$

leading to the following introduction and elimination rules

$$\begin{array}{c}
z \in \text{wp } U_1 \ C \vdash z \in \text{wp } U_0 \ C \\
U_0 \sqsupseteq_{wp} U_1
\end{array}$$

where $z$ and $C$ are fresh variables:

$$\begin{array}{c}
U_0 \sqsupseteq_{wp} U_1 \\
t \in \text{wp } U_1 \ C \\
t \in \text{wp } U_0 \ C
\end{array} \quad (\sqsupseteq_{wp})$$

The theory $\sqsupseteq_{wp}$ is also equivalent to $\sqsupseteq_w$ and $\sqsupseteq_s$. The proof of this, and a number of other approaches and analyses, can be found in [11].

### 5.3.2 Data Refinement

Data refinement is the more interesting and sophisticated paradigm. Formalising the usual approaches to forward and backward simulation in $\mathcal{Z}_C$ is straightforward.

We begin with the lifting of simulations:

$$S^{\lceil T_1 \lor T_0 \rceil} = \text{df } \{ z_1 \star z'_1 \in T_1 \star T_0 \mid z_1 \neq \bot \Rightarrow z_1 \star z'_1 \in S \}$$
leading to the following rules:

\[
\begin{align*}
  t_1 \cdot t_2' &\in T_1 \cdot T_{\omega_1}, \quad t_1 \not\perp &\quad \rightarrow \quad t_1 \cdot t_2' \in S & \\
  t_1 \cdot t_2' &\in S & \\
  t_1 \cdot t_2' &\in S & \\
  t_1 \cdot t_2' &\in S &
\end{align*}
\]

Then we can define WF\(_\star\)-refinement, a theory of forward simulation data refinement:

\[
U_0 \Downarrow_{w_f} U_1 \iff U_0 \Downarrow U_1 \Downarrow U_1 \Downarrow S
\]

leading to the following rules. Let \(z_0 \) and \(z_1 \) be fresh:

\[
\begin{align*}
  z_1 \cdot z_0' &\in S, \quad \text{Pre } U_1 z_1 \quad \text{by fresh variables:} \quad z_1 \cdot z_0' \in U_1 \Downarrow S \quad \text{by fresh variables:} \quad z_1 \cdot z_0' \in U_1 \Downarrow S \quad \text{by fresh variables:} \quad z_1 \cdot z_0' \in U_1 \Downarrow S \quad \text{by fresh variables:} \quad z_1 \cdot z_0' \in U_1 \Downarrow S
\end{align*}
\]

As with operation refinement, it is possible to define an equivalent theory (SF-refi-

\[
\begin{align*}
  \exists_{\text{sf}} &\quad U_0 \equiv_{\text{sf}} U_1 \quad \text{such that:} \quad U_0 \equiv_{\text{sf}} U_1 \quad \text{such that:} \quad U_0 \equiv_{\text{sf}} U_1 \quad \text{such that:} \quad U_0 \equiv_{\text{sf}} U_1 \quad \text{such that:} \quad U_0 \equiv_{\text{sf}} U_1
\end{align*}
\]

The usual side-conditions apply to the eigenvariable \(y\).

The theories \(\exists_{\text{sf}}\) and \(\exists_{\text{sf}}\) are equivalent [9].

A similar analysis for backwards refinement is also possible. Let \(x, x_0, x_1, z, z_0\) be fresh variables. Then SB-refinement is given by the following theory:

\[
\begin{align*}
  x \cdot z' &\in S \implies \text{Pre } U_1 z \quad \text{by fresh variables:} \quad x \cdot z' \in U_1 \Downarrow S \quad \text{by fresh variables:} \quad x \cdot z' \in U_1 \Downarrow S \quad \text{by fresh variables:} \quad x \cdot z' \in U_1 \Downarrow S
\end{align*}
\]
The usual side-conditions apply to the eigenvariable $y$.

WB•-refinement is: Let $z_0, z_1$ be fresh.

$$
\frac{z_0 \star z_1' \in \hat{U}_0 \Downarrow \hat{S} \vdash z_0 \star z_1' \in \hat{U}_1}{U_0 \Downarrow_{wbf} U_1} (\exists_{wbf})
$$

These two theories are also equivalent [8].

The weakest precondition approach can also be generalised to data refinement. For example, the following is a theory of weakest precondition data refinement (forwards case) for $Z$. First we need the image operator for simulations with respect to a (postcondition) set $C$:

$$
\left[ C^{\overrightarrow{T_1}} \right] S^{\overrightarrow{T_0 \overrightarrow{T_1}}} = \{ t_0 \in T_0 \mid \exists z_1 \in C \bullet z_0 \star z_1' \in S \}.
$$

This leads to the following theory, WPF-refinement:

$$
\frac{z \in \left[ wp U_1 C \right] S \vdash z \in \left[ wp U_0 [C] S' \right]}{U_0 \Downarrow_{wpf} U_1} (\exists_{wpf})
$$

where $z$ and $C$ are fresh variables.

$$
\frac{U_0 \Downarrow_{wpf} U_1 \quad t \in \left[ wp U_1 C \right] S}{t \in \left[ wp U_0 [C] S' \right]} (\exists_{wpf}^{-})
$$

The usual side-conditions apply to the eigenvariable $y$.

A weakest precondition data refinement for $Z$ in the backwards case is also possible. First we define the co-image of $S$ under the postcondition $C$ to be the set (of type $\mathbb{P} T_0$) of all concrete states, drawn from the domain of $S$, which only represent abstract states that are members of $C$.

$$
S^{\overrightarrow{T_0 \overrightarrow{T_1}}} \left[ C^{\overrightarrow{T_1}} \right] = \{ t_0 \in T_0 \mid \forall z_1 \bullet z_0 \star z_1' \in S \Rightarrow z_1 \in C \}.
$$

This leads to WPB-refinement:

$$
\frac{z \in S[wp U_1 C] \vdash z \in wp U_0 [C]}{U_0 \Downarrow_{wpb} U_1} (\exists_{wpb}^+)
$$

where $z$ and $C$ are fresh variables.

$$
\frac{U_0 \Downarrow_{wpb} U_1 \quad t \in S[wp U_1 C]}{t \in wp U_0 [C]} (\exists_{wpb}^-).
$$
The theories \( \sqsubseteq_{wpf} \) and \( \sqsubseteq_{wpb} \) are equivalent to (and undoubtedly simpler than) \( \sqsubseteq_{sf} \) and \( \sqsubseteq_{sb} \) (hence to \( \sqsubseteq_{wf} \) and \( \sqsubseteq_{wb} \)) respectively [10].

5.4 Monotonicity

It is perhaps rather strange that equality rather than refinement should be the fundamental relation of Z. It would be quite usual for a specification framework to take the latter as its fundamental relation. Equality would then appear as inter-refinability. As we have seen, there is a way in which refinement could be construed as more fundamental than equality: if we were content with partial correctness refinement where preconditions are firing conditions. But this is not at all satisfactory. The consequence, however, is that inter-refinability cannot be a finer relation than equality, and there is then a price to pay: the schema calculus is not monotonic with respect to refinement.

Monotonicity can to some extent be rehabilitated by imposing side-conditions on the way in which schema operators are used. For example, if we have

\[
\forall z \bullet \text{Pre } U_0 z \land \text{Pre } U_2 z \Rightarrow \text{Pre } (U_0 \land U_2) z
\]

then we also have

\[
\frac{U_0 \sqsubseteq_s U_1}{U_0 \land U_2 \sqsubseteq_s U_1 \land U_2}.
\]

In other words, schema conjunction is monotonic in such circumstances.

The logic proves to be a very useful tool in synthesizing such side-conditions, as we now illustrate.

For schema disjunction we require this sidecondition:

\[
\forall z \bullet \text{Pre } U_0 z \land \text{Pre } U_2 z \Rightarrow \text{Pre } U_1 z.
\]

Then the following rule is derivable:

\[
\frac{U_0 \sqsubseteq_s U_1}{U_0 \lor U_2 \sqsubseteq_s U_1 \lor U_2}.
\]

In this case let us consider the \( Z_C \) proof:

\[
\frac{U_0 \sqsubseteq_s U_1 \quad \text{Pre } U_1 z \quad \text{Pre } U_2 z \quad \text{Pre } U_0 z \lor U_2 z \quad \text{Pre } U_0 z \lor U_2 z \quad \text{Pre } U_1 z \lor U_2 z \quad \text{Pre } U_0 z \lor U_2 z}{U_0 \lor U_2 \sqsubseteq_s U_1 \lor U_2}.
\]
where $\delta_0$ stands for the following branch (where we write $z$ for $z_0 \ast z_1'$)

$$
\begin{align*}
\delta_1 & \vdash U_0 \models U_1 \quad \text{Pre } U_1 \models z_0 \quad \text{Pre } U_2 \models z_0 \quad \text{(3)} \\
& \quad \quad \frac{z \in U_1 \lor U_2}{z \in U_1 \lor U_2} \quad \text{(3)} \\
\end{align*}
$$

and $\delta_1$ is

$$
\begin{align*}
\frac{z_0 \ast z_1' \in U_0}{\text{Pre } U_0 \models z_0} \quad \frac{\text{Pre } U_2 \models z_0}{\text{Pre } U_0 \models z_0 \land \text{Pre } U_2 \models z_0} \quad \text{(4)} \\
\end{align*}
$$

Note the point (in the right-most sub-proof of $\delta_1$) where the side-condition is required. A proof attempt without the side-condition in place fails at this point. But the available assumptions and the required conclusion immediately articulate the minimum condition for the result to hold.

For a comprehensive investigation of the question of the monotonicity of the schema operators with respect to refinement, see [12].

6 CONCLUSIONS

This paper addresses two aims: first, to provide an accessible introduction to the Z logic based on ZC, and second, to survey a range of more advanced applications of this logic with references to the relevant literature.

The reader will have noticed one or two occasions on which concepts here differ from vernacular Z (and indeed ISO Z). It is worth reflecting a little on the reasons for these differences. Z was not originally introduced as a theory, rather as a notation or language. The early formal work on Z concentrated on semantics (see [28] in particular) with logic making an appearance somewhat later (see [32] in particular, and also [13, 23, 34, 4, 5, 7, 15, 24, 6] for other developments and approaches). The emphasis on semantics did not naturally lead to an increase in the level of formality for future investigations: a logic permits direct reasoning in the language, whereas reasoning in the semantics is hardly a practical (nor even a desirable) matter. It should not be too surprising therefore to discover opportunities and difficulties when a language, which has to a great extent developed independently of its mathematical foundations, is considered in a logical context. These tensions are very much a part of previous work to which we have already referred: whilst [19] is largely devoted
to vernacular Z, [17, 18] explicitly ask questions about vernacular Z which arise as a consequence of the logical analysis. In this paper, the deviations (apart from trivial notational differences) are modest but present: priming considered as a bijective operation between observations and co-observations (Section 3.1) and a hint in the direction of novelty in connection with \( \theta \)-terms of the form \( \theta S' \) (Section 2.2). The papers [17, 18] are more revisionary, as their titles suggest.

A second theme we wish to highlight concerns our survey of more advanced areas: the fact that the logic permits the formalisation of associated conceptual apparatus alongside the specification language itself. Of particular note is the wide variety of refinement theories we presented. In addition to the material discussed in this paper, it is also possible to formalise programming notations within the logic and relationships between programs and specifications. This is covered in [20] and in [21]; again all the formalisation and analysis takes place within the single framework.

Finally, note that now we have a Z logic, we can use it to give logics (via definitions and hence derived rules) to various other formalisms. One formalism that we have investigated is the Statechart-like \( \mu \)-charts [14]. Once the definitions that formalise them have been made, and the Z logic rules are expressed via the definitions as \( \mu \)-chart rules, all the paraphernalia that exist to support Z can be used to support them. Proof tools are obvious examples of this, but also, and more interestingly, the very refinement rules that we presented in Section 5 can be used to derive a theory of refinement for \( \mu \)-charts.

As a parting thought, providing a common logic for Z that all tool builders can use as a standard is an obvious outcome of the work presented here, though how many tools will be checked and, if necessary, updated to conform to the Z logic remains to be seen.

Acknowledgements

Martin Henson is especially grateful to the late Peter Wexler, whose intellectual judgement and clarity of thought were a beacon for all who knew him.

Our work owes much to an earlier attempt at a Z logic by Ray Turner. Much of the work briefly surveyed in Section 5 is joint work with Moshe Deutsch. We wish to thank the anonymous referees for their useful comments and careful technical proofreading.

We have discussed Z, Z logic, and associated theories of refinement and program development, with more people than we can possibly list without running a risk of serious omission. There is no excuse, however, for failing to mention: Rob Arthan, Eerke Boiten, John Derrick, Moshe Deutsch, Lindsay Groves, Greg Reeve, Ray Turner, Mark Utting and Jim Woodcock.
REFERENCES

Martin Henson is currently Head of the Department of Computer Science at the University of Essex, U.K. He joined the Department in 1983 and for several years undertook research in the area of Functional Programming, with a special emphasis on verification and transformation. He wrote an early book in this area, with an emphasis on these topics, in 1987. During the first half of the next decade he published widely in the area of applications of constructive mathematical methods for software science, with an emphasis on specification and program derivation. This work was based on Feferman-style theories designed expressly for the purpose and was supported by EPSRC funding. Limitations of these theories for expressing program derivations led him to explore issues in “vernacular reasoning”: investigating the relationship between concise informal and verbose formalised arguments. Other limitations, in the area of specification, resulted in an interest in specification languages such as Z. After early attempts to constructivise Z, again with EPSRC funding, his joint work with Steve Reeves at Waikato University, NZ, on Z logic and (classical) logic-based program derivation from Z specifications, and his more recent work with Moshe Deutsch at the University of Essex, on a logical analysis of refinement, have resulted in the range of results and approaches summarised in this paper.

Steve Reeves has received his undergraduate degree in mathematics at University of Birmingham, U.K., and PhD in computer science (automated theorem-proving) at University of Birmingham, U.K. He then moved to University of Essex for one year, then to University of London (QMC) for 11 years, and then to University of Waikato, New Zealand, where he has been for the last nine years. He works on logics for specification languages: Z, micro-charts (a clean and clear version of Statecharts), and putting together state- and process-based languages.
Jonathan Bowen is professor of computing at London South Bank University where he heads the Centre for Applied Formal Methods. From 1995 to March 2000, he was a lecturer at the Department of Computer Science, The University of Reading, where he led the Formal Methods and Software Engineering Group. Previously he was a senior researcher at the Oxford University Computing Laboratory Programming Research Group where he worked under the guidance of Sir Tony Hoare, FRS. Between 1979 and 1984 he worked at Imperial College, London as a research assistant, latterly in the interdepartmental Wolfson Microprocessor Laboratory. He has been involved with the field of computing in both industry (including Marconi Instruments, Logica and Silicon Graphics Inc.) and academia since 1977. His interests include formal methods, safety-critical systems, the Z notation, provably correct systems, rapid prototyping using logic programming, decompilation, hardware compilation, software/hardware co-design, the history of computing and on-line museums. He holds an MA degree in Engineering Science from Oxford University. He won the 1994 IEE Charles Babbage Premium award and managed the ESPRIT ProCoS-WG Working Group of 25 European partners (1993–1997) on Provably Correct Systems. During 1999 he was a Visiting Research Fellow at the United Nations University International Institute for Software Technology (UNU/IIST), Macau. He has produced over 200 publications and 13 books, and has served on about 50 programme committees. He is the Chair of the Z User Group, Chair of the BCS Formal Aspects of Computing Science Specialist Group, and a member of the IEEE Computer Society and the ACM.