Computing and Informatics, Vol. 22, 2003, 381–415

Z LOGIC AND ITS CONSEQUENCES

Martin C. Henson

Department of Computer Science, University of Essex, U. K.
e-mail: hensm@essex.ac.uk
www: http://cswww.essex.ac.uk/staff/henson

Steve Reeves

Department of Computer Science, University of Waikato, N. Z.
e-mail: stever@cs.waikato.ac.nz
www: http://www.cs.waikato.ac.nz/Staff/steve-reeves.html

Jonathan P. Bowen

Centre for Applied Formal Methods, London South Bank University, U. K.
e-mail: jonathan.bowen@lsbu.ac.uk
www: http://www.jpbowen.com/

For Peter Wexler – *in memoriam*

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 Z[2] 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 vernacular[3] 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 $\mathcal{Z}_\mathcal{C}$) 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 encyclopædic 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 $\mathcal{Z}_\mathcal{C}$ 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 $\mathcal{Z}_{\mathcal{C}}$ 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:

$$\begin{array}{|l}
\textit{Jug} \\
\hline
\textit{volume} : \mathbb{N} \\
\textit{temp} : \mathbb{N} \\
\hline
\textit{volume} \leq 250 \\
\textit{temp} \leq 100 \\
\hline
\end{array}$$

Written in linear form this would be:

$$\textit{Jug} \mathrel{\widehat=} [\textit{volume} : \mathbb{N};\ \textit{temp} : \mathbb{N} \mid \textit{volume} \leq 250 \wedge \textit{temp} \leq 100]$$

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.[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)

[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.[5]

We will write bindings like this:[6]

$$\langle\!| \; \mathit{volume} \Rrightarrow n, \mathit{temp} \Rrightarrow m \; |\!\rangle$$

where, in this case, $n \in \mathbb{N}$ etc. Naturally, it *should* follow that, for example:

$$\langle\!| \; \mathit{volume} \Rrightarrow 100, \mathit{temp} \Rrightarrow 20 \; |\!\rangle \in \mathit{Jug}$$

and also:

$$\langle\!| \; \mathit{volume} \Rrightarrow 100, \mathit{temp} \Rrightarrow 200 \; |\!\rangle \notin \mathit{Jug}.$$

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:

$$\langle\!| \; \mathit{volume} \Rrightarrow 100, \mathit{temp} \Rrightarrow 20 \; |\!\rangle.\mathit{volume} = 100.$$

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

$$\left[\cdots \; \mathtt{z}_i^{T_i} \; \cdots \right].$$

This is an unordered sequence of typed (indicated by superscripts) observations (the $\mathtt{z}_i$). Then *schemas* are either *schema sets*:

$$\left[\cdots \; \mathtt{z}_i : C_i^{\mathbb{P}\, T_i} \; \cdots \right]$$

or they are *atomic schemas*:

$$\left[S \mid P \right]$$

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:

$$\mathbb{N} =_{df} \{ z^{\mathbb{N}} \mid \mathit{true} \}$$

under the cartesian product, power type and schema type operations.[7]

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 $\mathcal{Z_C}$, for sets, we have:

$$\frac{P[z/t]}{t \in \{z \mid P\}} \; (\{\}^+) \qquad \frac{t \in \{z \mid P\}}{P[z/t]} \; (\{\}^-).$$

[5] See for example [33]. In chapter 11, page 149, they are “variables”; by page 154 they are “components” (constants).

[6] ISO Z uses == rather than $\Rrightarrow$, a notation which dates back to [28] and [29].

[7] In fact $\mathbb{N}$ is only one possible base type. See Section 3 for further details.

Note that $\mathcal{Z}_{\mathcal{C}}$ is strongly typed, so these (typed) set comprehensions present no technical difficulties. See Section 3 for further details.

For bindings, $\mathcal{Z}_{\mathcal{C}}$ has:

$$\frac{}{\langle\!| \cdots \mathtt{z}_i \Rrightarrow t_i \cdots |\!\rangle.\mathtt{z}_i = t_i}\ (\Rrightarrow^{=}_{0}) \qquad \frac{}{\langle\!| \cdots \mathtt{z}_i \Rrightarrow t.\mathtt{z}_i \cdots |\!\rangle = t^{[\cdots \mathtt{z}_{\mathtt{i}}^{\mathtt{T_i}} \cdots]}}\ (\Rrightarrow^{=}_{1})$$

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 $\mathcal{Z}_{\mathcal{C}}$ definitions:

$$\left[\cdots\ \mathtt{z}_i : C_i \cdots\right] =_{df} \{x \mid \cdots \wedge x.\mathtt{z}_i \in C_i \wedge \cdots\}$$

and

$$\left[\, S \mid P \,\right] =_{df} \{z \in S \mid z.P\}.$$

The *binding selection* operator, introduced in the object logic for selection from bindings (that is, $\mathcal{Z}_{\mathcal{C}}$ terms such as $z.\mathtt{x}$) is generalised into a meta-language substitution over terms (that is, meta-terms such as $z.\mathrm{t}$) and over propositions (meta-terms such as $z.\mathrm{P}$)[8]. 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 $\mathcal{Z}_{\mathcal{C}}$:

$$\frac{\cdots \quad t_i \in C_i \quad \cdots}{\langle\!| \cdots \mathtt{z}_i \Rrightarrow t_i \cdots |\!\rangle \in [\cdots \mathtt{z}_i : C_i \cdots]}\ ([]^{+}) \qquad \frac{t \in [\cdots \mathtt{z}_i : C_i \cdots]}{t.\mathtt{z}_i \in C_i}\ ([]^{-})$$

and, for *atomic schemas*:

$$\frac{t \in S \quad t.P}{t \in [S \mid P]}\ (S^{+}) \qquad \frac{t \in [S \mid P]}{t \in S}\ (S^{-}_{0}) \qquad \frac{t \in [S \mid P]}{t.P}\ (S^{-}_{1}).$$

Then for example, writing b for $\langle\!|\ volume \Rrightarrow 100, temp \Rrightarrow 20\ |\!\rangle$, we have:

$$\dfrac{\dfrac{\begin{matrix}\vdots \\ 100 \in \mathbb{N} \wedge 20 \in \mathbb{N}\end{matrix}}{b \in [volume : \mathbb{N}, temp : \mathbb{N}]}\ ([]^{+}) \qquad \begin{matrix}\vdots \\ 100 \leq 250 \wedge 20 \leq 100\end{matrix}}{b \in Jug}\ (S^{+})$$

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

[8] This is modelled to some extent on the more complex *object language* substitution *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:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{b \in \mathit{Jug}}\ 1,(S_1^-)}{b.\mathit{volume} \leq 250 \wedge b.\mathit{temp} \leq 100}}{b.\mathit{volume} * b.\mathit{temp} \leq 25000}}{\forall\, b \in \mathit{Jug} \bullet b.\mathit{volume} * b.\mathit{temp} \leq 25000}\ 1.$$

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:

$$\mathit{Jug} \wedge \mathit{Jug}'.$$

This is also often referred to as $\Delta \mathit{Jug}$ 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 $S_0 \wedge S_1$ 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 $S_0 \wedge S_1$ to be well-defined, these types must *agree on their overlap.* We will write $T_0 \curlyvee T_1$ (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 $S_0 \wedge S_1$ will be *all* the bindings z in $T_0 \curlyvee T_1$ 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 $z \upharpoonright T$ 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:

$$\langle\!|\ x \Rrightarrow 3, y \Rrightarrow 4\ |\!\rangle \upharpoonright [x^{\mathbb{N}}] = \langle\!|\ x \Rrightarrow 3\ |\!\rangle.$$

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:

$$\frac{t^{T_0}.\mathsf{z}_i = t_i}{(t \upharpoonright T_1).\mathsf{z}_i = t_i}\ (\upharpoonright^{=}) \qquad T_1 \preceq T_0 \text{ and } \mathsf{z} \in \alpha T_1.$$

The meta-term αT 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 $T_1 \preceq T_0$:

$$z^{T_0} \mathrel{\dot{\in}} S^{\mathbb{P}\, T_1} =_{df} z \upharpoonright T_1 \in S.$$

This idea can also be applied to equality:

$$t_0^{T_0} \;\doteq\; t_1^{T_1} =_{df} t_0 \upharpoonright (T_0 \curlywedge T_1) = t_1 \upharpoonright (T_0 \curlywedge T_1).$$

Here we have written $T_0 \curlywedge T_1$ for schema type intersection. The notation is most usefully employed when $T_1 \preceq T_0$ or $T_0 \preceq T_1$.

More generally we have:

$$t_0^{T_0} =_T t_1^{T_1} =_{df} t_0 \upharpoonright T = t_1 \upharpoonright T.$$

This notation is most usefully employed when $T \preceq T_0$ *and* $T \preceq T_1$.

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

$$S_0^{\mathbb{P}\,T_0} \wedge S_1^{\mathbb{P}\,T_1} =_{df} \{z^{\mathbb{P}(T_0 \curlyvee T_1)} \mid z \upharpoonright T_0 \in S_0 \wedge z \upharpoonright T_1 \in S_1\}$$

which leads immediately to the following rules:

$$\frac{t \;\dot\in\; S_0 \quad t \;\dot\in\; S_1}{t \in S_0 \wedge S_1}\,(S^+_\wedge) \qquad \frac{t \in S_0 \wedge S_1}{t \;\dot\in\; S_0}\,(S^-_{\wedge_0}) \qquad \frac{t \in S_0 \wedge S_1}{t \;\dot\in\; S_1}\,(S^-_{\wedge_1}).$$

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

AddWater

ΔJug
$more? : [v : \mathbb{N}, t : \mathbb{N}]$

$volume' = volume + more?.v$
$temp' = (volume * temp + more?.v * more?.t) \text{ div } volume'$

The declaration in this case amounts to the schema:

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

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:

$$\langle\!| \; volume \Rrightarrow 50, temp \Rrightarrow 25, more? \Rrightarrow m, volume' \Rrightarrow 150, temp' \Rrightarrow 41 \; |\!\rangle$$

and m is the binding:

$$\langle\!| \; v \Rrightarrow 100, t \Rrightarrow 50 \; |\!\rangle.$$

We have:

$$\dfrac{\dfrac{\dfrac{\begin{matrix}\vdots\\ b \mathrel{\dot{\in}} Jug\end{matrix} \quad \begin{matrix}\vdots\\ b \mathrel{\dot{\in}} Jug'\end{matrix}}{b \mathrel{\dot{\in}} Jug \wedge Jug'}\,(S^+_\wedge) \quad \begin{matrix}\delta\\ \vdots\\ b \mathrel{\dot{\in}} [more? : [v : \mathbb{N}, t : \mathbb{N}]]\end{matrix}}{b \in Jug \wedge Jug' \wedge [more? : [v : \mathbb{N}, t : \mathbb{N}]]}\,(S^+_\wedge) \quad \begin{matrix}\vdots\\ P\end{matrix}}{b \in AddWater}\,(S^+)$$

writing P for $150 = 50 + 100 \wedge 41 = (50 * 25 + 100 * 50)\ \mathrm{div}\ 150$ and where, for example, δ is:

$$\dfrac{\dfrac{}{b \doteq \langle\!|\ more?\Rrightarrow m\ |\!\rangle} \quad \dfrac{\dfrac{\dfrac{}{100 \in \mathbb{N}} \quad \dfrac{}{150 \in \mathbb{N}}}{m \in [v : \mathbb{N}, t : \mathbb{N}]}}{\langle\!|\ more?\Rrightarrow m\ |\!\rangle \in [more? : [v : \mathbb{N}, t : \mathbb{N}]]}}{b \mathrel{\dot{\in}} [more? : [v : \mathbb{N}, t : \mathbb{N}]]}.$$

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

$$\begin{array}{|l}
\multicolumn{1}{l}{TakeTemp}\\ \hline
\Xi Jug\\
read! : \mathbb{N}\\ \hline
read! = temp\\ \hline
\end{array}$$

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

$$\begin{array}{|l}
\multicolumn{1}{l}{TakeTemp}\\ \hline
\Delta Jug\\
read! : \mathbb{N}\\ \hline
read! = temp\\
\theta Jug = \theta Jug'\\ \hline
\end{array}$$

According to the definition given above, this is interpreted as the following set of bindings in $\mathcal{Z}_{\mathcal{C}}$:

$$\{z \in \Delta Jug \wedge [more? : [v : \mathbb{N}, t : \mathbb{N}] \wedge [read! : \mathbb{N}]] \mid z.(read! = temp \wedge \theta Jug = \theta Jug')\}.$$

What is so far missing from our account is an explanation of θ-terms. In the *unprimed* case:

$$\theta S^{\mathbb{P}[\cdots \mathtt{z}_i^{\mathtt{T}_i} \cdots]} =_{df} \langle\!|\ \cdots\ \mathtt{z}_i \Rrightarrow \mathtt{z}_i\ \cdots\ |\!\rangle.$$

Thus $z^{T_0}.\theta S^{\mathbb{P}\,T_1} = z \upharpoonright T_1$ whenever $T_1 \preceq T_0$.

In the *primed* case we have $\theta S' = \theta' S$ where:

$$\theta' S^{\mathbb{P}[\cdots \mathtt{z}_i^{\mathtt{T}_i} \cdots]} =_{df} \langle\!|\ \cdots\ \mathtt{z}_i \Rrightarrow \mathtt{z}_i'\ \cdots\ |\!\rangle.$$

The second of these suggests, correctly, that in fact we have an operation (called θ') on S rather than S'. Indeed, we have not provided a precise explanation of the priming of schemas: θ' is the more fundamental concept:

$$\left[\cdots \mathtt{x}_i : T_i \cdots\right]' =_{df} \left[\cdots \mathtt{x}'_i : T_i \cdots\right]$$

and:

$$\left[\, S \mid P \,\right]' =_{df} \left[\, S' \mid \theta' S.P \,\right]$$

The special Z term θ 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 $(\Rrightarrow^{=}_{1})$ (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 $\mathcal{Z}_C$ and that the rules for reasoning about such schema expressions are:

$$\frac{t \in S}{t \mathrel{\dot{\in}} \exists \mathbf{z} \in T \bullet S}\ (S^{+}_{\exists})$$

$$\frac{t \in \exists \mathbf{z} \in T \bullet S \quad y \in S, y \doteq t \vdash P}{P}\ (S^{-}_{\exists}).$$

Let us illustrate promotion by examining the simplest of examples.

Example 5. Consider the following trivial operation:

Inc

$v, v' : \mathbb{N}$

$v' = v + 1$

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:

$$\begin{array}{|l}
\hline
\multicolumn{1}{l}{\Phi Pair} \\
v, v' : \mathbb{N} \\
w, w' : \mathbb{N} \times \mathbb{N} \\
\hline
w.1 = v \\
w'.1 = v' \\
w'.2 = w.2 \\
\hline
\end{array}$$

and the global operation is:

$$PairInc \mathrel{\widehat{=}} \exists\, v, v' : \mathbb{N} \bullet Inc \wedge \Phi Pair.$$

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

$$\langle\!|\ w \Rrightarrow (3,5), w' \Rrightarrow (4,5)\ |\!\rangle \in PairInc.$$

We will write this binding as b_0 and the extended binding

$$\langle\!|\ v \Rrightarrow 3, v' \Rrightarrow 4, w \Rrightarrow (3,5), w' \Rrightarrow (4,5)\ |\!\rangle$$

as b_1. This is straightforward:

$$\dfrac{\begin{array}{c}\vdots\\ b_0 \doteq b_1\end{array} \qquad \dfrac{\dfrac{\begin{array}{c}\delta_0\\ \vdots\\ b_1 \mathrel{\dot{\in}} Inc\end{array} \quad \begin{array}{c}\delta_1\\ \vdots\\ b_1 \mathrel{\dot{\in}} \Phi Pair\end{array}}{b_1 \in Inc \wedge \Phi Pair}\,(S^+_{\wedge})}{b_1 \mathrel{\dot{\in}} PairInc}\,(S^+_{\exists})}{b_0 \in PairInc} \qquad .$$

Let b_2 be $\langle\!|\ x \Rrightarrow 3, x' \Rrightarrow 4\ |\!\rangle$, then δ_0 is

$$\dfrac{\begin{array}{c}\vdots\\ b_1 \doteq b_2\end{array} \qquad \dfrac{\dfrac{\overline{3 \in \mathbb{N}} \quad \overline{4 \in \mathbb{N}}}{b_2 \in [v, v' : \mathbb{N}]} \quad \overline{4 = 3 + 1}}{b_2 \in Inc}}{b_1 \mathrel{\dot{\in}} Inc}$$

and δ_1 is

$$\dfrac{\begin{array}{c}\vdots\\ b_1 \in [v, v' : \mathbb{N}, w, w' : \mathbb{N} \times \mathbb{N}]\end{array} \quad \begin{array}{c}\vdots\\ 3 = 3 \wedge 4 = 4 \wedge 5 = 5\end{array}}{b_1 \mathrel{\dot{\in}} \Phi Pair}\,.$$

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

$$\forall\, b \in \mathit{PairInc} \bullet b.w.2 = b.w'.2$$

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

$$\dfrac{\dfrac{\dfrac{}{b \in \mathit{PairInc}}\,1 \qquad \dfrac{\dfrac{}{y \doteq b}\,2 \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{y \in \mathit{Inc} \wedge \Phi \mathit{Pair}}\,2}{y \mathrel{\dot{\in}} \Phi \mathit{Pair}}}{y.w.1 = y.v \wedge t.w'.1 = y.v' \wedge y.w.2 = y.w'.2}}{y.w.2 = y.w'.2}}{b.w.2 = b.w'.2}}{b.w.2 = b.w'.2}\,2, (S^{-}_{\exists})}{\forall\, b \in \mathit{PairInc} \bullet b.w.2 = b.w'.2}\,1$$

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:

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

Types of the form Υ are the names of *free types* and are given by equations of the form

$$\Upsilon \ ::= \ \cdots \mid \ c_i \ \langle\!\langle \cdots \Upsilon_{ij} \cdots \rangle\!\rangle \mid \cdots$$

where any of the Υ_{ij} may be Υ (permitting recursion). In particular, $\langle\!\langle \cdots \Upsilon_{ij} \cdots \rangle\!\rangle$ may be omitted. An important example is

$$\mathbb{N} \ ::= \ \mathit{zero} \mid \ \mathit{succ} \ \langle\!\langle \mathbb{N} \rangle\!\rangle.$$

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.[10]

Types of the form $[\cdots \mathsf{z}_i^{T_i} \cdots]$ (the order is not important) are called *schema types.* We write $\alpha[\cdots \mathsf{z}_i^{T_i} \cdots]$ for the alphabet set (in the meta-language) of observations $\{\cdots \mathsf{z}_i \cdots\}$. No observation may occur more than once in such a type. The symbols $\preceq$, $\curlywedge$, $\curlyvee$ and $-$ denote the *schema subtype* relation, and the operations of *schema type intersection*, *schema type union* and *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 *compatible* (common observations agree on their type).

The last important operation on types is *priming.* First we associate with every observation z its *co-observation* z' where $\mathsf{z}'' = \mathsf{z}$. Then we set $[\cdots \mathsf{z} \cdots]'$ to be $[\cdots \mathsf{z}' \cdots]$. 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.[11]

All further syntactic categories of the language of $\mathcal{Z}_{\mathcal{C}}$ 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 $\mathcal{Z}_{\mathcal{C}}$ have the form $\Gamma \vdash P$ where Γ is a set of formulæ. The logic is presented as a natural deduction system *in sequent form.* We shall omit all data (entailment symbol, contexts, type etc.) which remain unchanged by any rule.

3.2 The Terms of $\mathcal{Z}_{\mathcal{C}}$

First we have variables, bindings, pairs and their projections:[12]

$$\begin{array}{lcl} t^{T} & ::= & x^{T} \mid t^{[\cdots \mathsf{z}^{\mathsf{T}} \cdots]}.\mathsf{z} \mid t^{T \times T_1}.1 \mid t^{T_0 \times T}.2 \\ t^{T_0 \times T_1} & ::= & (t^{T_0}, t^{T_1}) \\ t^{[\cdots \mathsf{z}^{\mathsf{T}} \cdots]} & ::= & \langle\!| \cdots \ \mathsf{z} \Rrightarrow t^{T} \ \cdots |\!\rangle \end{array}$$

[10] For the reader interested in pursuing the technical issues concerning free-types, see [2, 30] for example.

[11] 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.

[12] The reader may already have noticed, from examples in Section 2, that we carefully distinguish *observation meta-variables* and *variable meta-variables.* In the *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.

These terms are characterised by various logical rules:

$$\frac{}{\langle\!| \cdots \mathtt{z}_i \Rrightarrow t_i \cdots |\!\rangle.\mathtt{z}_i = t_i}\ (\Rrightarrow_{\mathtt{0}}^{=}) \qquad \frac{}{\langle\!| \cdots \mathtt{z}_i \Rrightarrow t.\mathtt{z}_i \cdots |\!\rangle = t^{[\cdots \mathtt{z}_{\mathtt{i}}^{\mathtt{T_i}} \cdots]}}\ (\Rrightarrow_{\mathtt{1}}^{=})$$

$$\frac{}{(t_0, t_1).1 = t_0}\ (()_{\mathtt{0}}^{=}) \qquad \frac{}{(t_0, t_1).2 = t_1}\ (()_{\mathtt{1}}^{=}) \qquad \frac{}{(t.1, t.2) = t}\ (()_{\mathtt{2}}^{=}).$$

Second, we have the filtered (restricted) bindings.

$$t^{T_0} \ ::= \ t^{T_1} \upharpoonright T_0 \quad \text{where } T_0 \preceq T_1$$

As we have seen, the rule for these is:

$$\frac{t^{T_0}.\mathtt{z}_i = t_i}{(t \upharpoonright T_1).\mathtt{z}_i = t_i}\ (\upharpoonright^{=}) \qquad T_1 \preceq T_0 \text{ and } \mathtt{z} \in \alpha\, T_1.$$

Third, there are the values of free-type:

$$t^{\Upsilon} \ ::= \ c_i \cdots t^{\Upsilon_{ij}} \cdots$$

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

$$\frac{\cdots z_{ij} \in \Upsilon_{ij} \cdots}{c_i \cdots z_{ij} \cdots \in \Upsilon}\ (\Upsilon^{+}) \qquad \frac{\cdots z_{ij} \in \Upsilon_{ij} \cdots \quad \cdots z_{kl} \in \Upsilon_{kl} \cdots}{c_i \cdots z_{ij} \cdots \neq c_k \cdots z_{kl} \cdots}\ (\Upsilon_{\neq})$$

$$\frac{c_i \cdots z_{ij} \cdots = c_i \cdots y_{ij} \cdots}{z_{ij} = y_{ij}}\ (\Upsilon_{=})$$

$$\frac{\cdots \quad \cdots z_{ij} \in \Upsilon_{ij} \cdots, \cdots P[z/y_k] \cdots \ \vdash \ P[z/c_i \cdots z_{ij} \cdots] \quad \cdots}{z \in \Upsilon \ \vdash \ P}\ (\Upsilon^{-})$$

where the y_k are all those variables occurring in the z_{ij} with type Υ.

Finally, we have sets:

$$t^{\mathbb{P}\,T} \ ::= \ \{z^T \mid P\}.$$

These are governed by:

$$\frac{P[z/t]}{t \in \{z \mid P\}}\ (\{\}^{+}) \qquad \frac{t \in \{z \mid P\}}{P[z/t]}\ (\{\}^{-}).$$

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*.

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

$$t.P = P[\cdots \mathsf{z}_i \cdots / \cdots t.\mathsf{z}_i \cdots].$$

3.3 The Formulæ of $\mathcal{Z_C}$

The formulæ of $\mathcal{Z_C}$ delineate a typed bounded predicate logic.

$$P \ ::= \ \mathit{false} \ \mid \ t^T = t^T \ \mid \ t^T \in C^{\mathbb{P}\,T} \ \mid \ \neg P \ \mid \ P \vee P \ \mid \ \exists\, z^T \in C^{\mathbb{P}\,T} \bullet P$$

The logic of $\mathcal{Z_C}$ is classical, so the remaining logical operations are available by definition. We also, as usual, abbreviate $\neg\,(t \in C)$ to $t \notin C$.

A crucial observation is *unicity of types*: every term of $\mathcal{Z_C}$ 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:

$$\frac{P_0}{P_0 \vee P_1}\,(\vee_0^+) \qquad \frac{P_1}{P_0 \vee P_1}\,(\vee_1^+) \qquad \frac{P_0 \vee P_1 \quad P_0 \ \vdash \ P_2 \quad P_1 \ \vdash \ P_2}{P_2}\,(\vee^-)$$

$$\frac{P \ \vdash \ \mathit{false}}{\neg P}\,(\neg^+) \qquad \frac{P \quad \neg P}{\mathit{false}}\,(false^+) \qquad \frac{\neg\neg P}{P}\,(\neg^-) \qquad \frac{\mathit{false}}{P}\,(false^-)$$

$$\frac{P[z/t] \quad t \in C}{\exists\, z \in C \bullet P}\,(\exists^+) \qquad \frac{\exists\, z \in C \bullet P_0 \quad y \in C, P_0[z/y] \ \vdash \ P_1}{P_1}\,(\exists^-).$$

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

$$\frac{}{\Gamma, P \ \vdash \ P}\,(ass) \qquad \frac{}{t = t}\,(ref) \qquad \frac{t_0 = t_1 \quad P[z/t_0]}{P[z/t_1]}\,(sub)$$

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

where:

$$t_0 \equiv t_1 =_{df} \forall\, z \in t_0 \bullet z \in t_1 \wedge \forall\, z \in t_1 \bullet z \in t_0.$$

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

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

$$\frac{}{\mathit{zero} \in \mathbb{N}} \qquad \frac{n \in \mathbb{N}}{\mathit{succ}\ n \in \mathbb{N}} \qquad \frac{n \in \mathbb{N}}{\mathit{zero} \neq \mathit{succ}\ n}$$

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

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

$$\frac{\Gamma \ \vdash \ P_1}{\Gamma, P_0 \ \vdash \ P_1}\ (wk).$$

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

$$\frac{}{t^T \in T}.$$

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.[13] We can define a *dependent function space* which is suitable for our purposes as follows:

$$\Pi_{(i \in I)}.B(i) =_{df} \{f \in I \to U \mid (\forall\, i \in I)(f(i) \in B(i))\}.$$

This we can harness to interpret the schema types of $\mathcal{Z_C}$:

$$\llbracket [\cdots \mathsf{z}_i^{T_i} \cdots] \rrbracket =_{df} \Pi_{(x \in I)}.B(x)$$

where $I =_{df} \{\cdots \mathsf{z}_i \cdots\}$ and $B(\mathsf{z}_i) =_{df} \llbracket T_i \rrbracket$. 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:

$$\begin{array}{ll} \llbracket \langle\!| \cdots \mathsf{z}_i \Rrightarrow t_i \cdots |\!\rangle \rrbracket & =_{df} \ f_0 \\ \llbracket t.\mathsf{z} \rrbracket & =_{df} \ \llbracket t \rrbracket (\mathsf{z}) \\ \llbracket t \upharpoonright T \rrbracket & =_{df} \ f_1 \end{array}$$

where $f_0 \in \llbracket [\cdots \mathsf{z}_i^{T_i} \cdots] \rrbracket$, $f_0(\mathsf{z}_i) = \llbracket t_i \rrbracket$, $f_1 \in \llbracket T \rrbracket$ and $f_1(\mathsf{z}) = \llbracket t \rrbracket (\mathsf{z})$ when $\mathsf{z} \in \alpha[D]$.

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

[13] $F(\omega)$ is a suitable universe: see [17] for further details.

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 $\Gamma \rhd P\ \mathit{prop}$ and $\Gamma \rhd t : T$. There are then typing rules such as:

$$\frac{t_0 : T \quad t_1 : T}{t_0 = t_1\ \mathit{prop}}\ (C_=) \qquad \frac{t : T \quad C : \mathbb{P}\, T}{t \in C\ \mathit{prop}}\ (C_\in)$$

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:

$$\frac{P_0\ \mathit{prop} \quad P_1\ \mathit{prop}}{P_0 \vee P_1\ \mathit{prop}}\ (C_\vee) \qquad \frac{x : T \rhd P\ \mathit{prop}}{\exists\, x : T \bullet P\ \mathit{prop}}\ (C_\exists).$$

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

$$\frac{\Gamma \vdash P_0 \quad \Gamma^- \rhd P_1\ \mathit{prop}}{\Gamma \vdash P_0 \vee P_1}\ (\vee_o^+)$$

and:

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

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

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

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

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 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 $T = [\cdots\ \mathsf{z}_i^{T_i}\ \cdots]$. The syntax of basic schemas is:

$$S^{\mathbb{P}\,T}\ ::=\ [\cdots \mathsf{z}_i : C_i^{T_i} \cdots]\ \big|\ [S^{\mathbb{P}\,T} \mid P].$$

These are the *schema sets* and *atomic schemas*, respectively. As usual, we will write schemas of the form: $[[\cdots \mathsf{z}_i : C_i \cdots] \mid P]$ as $[\cdots \mathsf{z}_i : C_i \cdots \mid P]$. We allow the obvious generalisation of our alphabet operator to atomic state schemas and state schema sets: $\alpha[S \mid P] =_{df} \alpha S$ and $\alpha[\cdots \mathsf{z}_i : C_i^{T_i} \cdots] =_{df} \alpha[\cdots \mathsf{z}_i^{T_i} \cdots]$.

Then these two basic schemas can be interpreted in $\mathcal{Z_C}$ as follows:[14]

$$\big[\cdots\ \mathsf{z}_i : C_i \cdots\big] =_{df} \{x \mid \cdots \wedge x.\mathsf{z}_i \in C_i \wedge \cdots\}$$

and

$$\big[\,S \mid P\,\big] =_{df} \{z \in S \mid z.P\}.$$

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

$$\frac{\cdots\quad t_i \in C_i \quad \cdots}{\langle\!|\ \cdots \mathsf{z}_i \Rrightarrow t_i \cdots\ |\!\rangle \in [\cdots \mathsf{z}_i : C_i \cdots]}\ ([]^{+}) \qquad \frac{t \in [\cdots \mathsf{z}_i : C_i \cdots]}{t.\mathsf{z}_i \in C_i}\ ([]^{-})$$

and, for *atomic schemas*:

$$\frac{t \in S \quad t.P}{t \in [S \mid P]}\ (S^{+}) \qquad \frac{t \in [S \mid P]}{t \in S}\ (S_0^{-}) \qquad \frac{t \in [S \mid P]}{t.P}\ (S_1^{-}).$$

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:

Inc

$v, v' : \mathbb{N}$

$v' = v + 1$

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:

$$[\![[S \mid P]]\!] =_{df} \{z \in [\![S]\!] \mid z.P\}$$

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:

$$\{z^{[v^{\mathbb{N}}, v'^{\mathbb{N}}]} \mid z.(v' = v + 1)\}$$

which simplifies to:

$$\{z^{[v^{\mathbb{N}}, v'^{\mathbb{N}}]} \mid z.v' = z.v + 1\}.$$

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

4.2 θ-terms

The special Z term θ is interpreted as described in Section 2.2:

$$\theta S^{\mathbb{P}[\cdots \mathtt{z}_\mathtt{i}^{\mathtt{T}_\mathtt{i}} \cdots]} =_{df} \langle\!| \cdots \ \mathtt{z}_i \Rrightarrow \mathtt{z}_i \ \cdots |\!\rangle$$

In the *primed* case we have $\theta S' = \theta' S$ where:

$$\theta' S^{\mathbb{P}[\cdots \mathtt{z}_\mathtt{i}^{\mathtt{T}_\mathtt{i}} \cdots]} =_{df} \langle\!| \cdots \ \mathtt{z}_i \Rrightarrow \mathtt{z}'_i \ \cdots |\!\rangle.$$

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:

Example 8. Consider the following schemas:

Example

ΔS

$\theta S = \theta S'$

where:

S

$v : \mathbb{N}$

Under the interpretation we will have:

$$\{z^{[v^{\mathbb{N}}, v'^{\mathbb{N}}]} \mid z.(\theta S = \theta S')\},$$

and this will simplify to:

$$\{z^{[v^{\mathbb{N}}, v'^{\mathbb{N}}]} \mid z.v = z.v'\}$$

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 $S_0 \vee S_1$ has the type $\mathbb{P}(T_0 \curlyvee T_1)$.

The definition of schema disjunction in $\mathcal{Z_C}$ is:

$$S_0^{\mathbb{P}\,T_0} \vee S_1^{\mathbb{P}\,T_1} =_{df} \{z^{\mathbb{P}(T_0 \curlyvee T_1)} \mid z \mathrel{\dot{\in}} S_0 \vee z \mathrel{\dot{\in}} S_1\}.$$

This leads to the following rules:

$$\frac{t \ \dot{\in}\ S_0}{t \in S_0 \vee S_1}\,(S^+_{\vee_0}) \qquad \frac{t \ \dot{\in}\ S_1}{t \in S_0 \vee S_1}\,(S^+_{\vee_1})$$

$$\frac{t \in S_0 \vee S_1 \quad t \ \dot{\in}\ S_0 \ \vdash\ P \quad t \ \dot{\in}\ S_1 \ \vdash\ P}{P}\,(S^-_{\vee}).$$

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 $S_0 \wedge S_1$ has the type $\mathbb{P}(T_0 \curlyvee T_1)$.

The definition of schema conjunction in $\mathcal{Z_C}$ is, as we have seen:

$$S_0^{\mathbb{P}\,T_0} \wedge S_1^{\mathbb{P}\,T_1} =_{df} \{z^{\mathbb{P}(T_0 \curlyvee T_1)} \mid z \mathrel{\dot{\in}} S_0 \wedge z \mathrel{\dot{\in}} S_1\},$$

and the rules are:

$$\frac{t \ \dot{\in}\ S_0 \quad t \ \dot{\in}\ S_1}{t \in S_0 \wedge S_1}\,(S^+_{\wedge}) \qquad \frac{t \in S_0 \wedge S_1}{t \ \dot{\in}\ S_0}\,(S^-_{\wedge_0}) \qquad \frac{t \in S_0 \wedge S_1}{t \ \dot{\in}\ S_1}\,(S^-_{\wedge_1}).$$

4.5 Schema Negation

Schema negation is straightforward:

$$\neg S^{\mathbb{P}\,T} =_{df} \{z^T \mid z \notin S\},$$

and these rules follow:

$$\frac{t \notin S}{t \in \neg S}\,(S^+_{\neg}) \qquad \frac{t \in \neg S}{t \notin S}\,(S^-_{\neg})$$

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 $[\mathtt{z} : T;\ S \mid P]$ is just $[[\mathtt{z} : T] \wedge S \mid P]$ and so on.

4.7 Schema Existential Hiding

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

Existentially quantified schemas are interpreted in $\mathcal{Z_C}$ as follows:

$$\exists\, \mathsf{z} : T_0 \bullet S^{\mathbb{P}\, T_1} =_{df} \{x \in T_1 - [\mathsf{z}^{T_0}] \mid \exists\, y \in T_1 \bullet y \in S \wedge x = y \upharpoonright (T_1 - [\mathsf{z}^{T_0}])\};$$

then these logical rules follow:

$$\frac{t \in S}{t \;\dot{\in}\; \exists\, \mathsf{z} : T \bullet S}\ (S^{+}_{\exists})$$

$$\frac{t \in \exists\, \mathsf{z} : T \bullet S \quad y \in S, y \;\doteq\; t \;\vdash\; P}{P}\ (S^{-}_{\exists}).$$

4.8 Schema Composition

In this and the next section we will consider operation schema, that is, those schemas whose type is $\mathbb{P}\, T$ where T has the form $T^{in} \curlyvee T^{out'}$ 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:

$$C_0 \star C_1 =_{df} \{z_0 \star z_1 \mid z_0 \in C_0 \wedge z_1 \in C_1\}.$$

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 $T_0^{out} = T_1^{in}$. Then:

$$U_0^{T_0^{in} \curlyvee T_0^{out'}} \fatsemi U_1^{T_1^{in} \curlyvee T_1^{out'}} =_{df} \{(z_0 \star z'_1)^{T_0^{in} \curlyvee T_1^{out'}} \mid \exists\, y^{T_0^{out'}} \bullet z_0 \star y' \in U_0 \wedge y \star z'_1 \in U_1\}.$$

The rules are then:

$$\frac{t_0 \star t'_2 \in U_0 \quad t_2 \star t'_1 \in U_1}{t_0 \star t'_1 \in U_0 \fatsemi U_1}\ (U^{+}_{\fatsemi})$$

$$\frac{t_0 \star t'_1 \in U_0 \fatsemi U_1 \quad t_0 \star y' \in U_0, y \star t'_1 \in U_1 \vdash P}{P}\ (U^{-}_{\fatsemi}).$$

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 $T^{in} \preceq V$. Then:

$$Pre\ \ U\ \ x^{V} =_{df} \exists\, z \in U \bullet x =_{T^{in}} z.$$

This leads to the following rules:

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

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 $\{z^{T^{in}} \mid Pre\ \ U\ \ z\}$.

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:

$$[T_0 \mid P_0] \wedge [T_1 \mid P_1] = [T_0 \curlyvee T_1 \mid P_0 \wedge P_1].$$

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:

$$\dfrac{\dfrac{\dfrac{\dfrac{t \in \neg[T \mid P]}{t \notin [T \mid P]}\,(S_{\neg}^{-})}{\neg(t \in T \wedge t.P)}}{t \notin T \vee \neg t.P} \quad \dfrac{\dfrac{\dfrac{}{t \notin T}\,(1) \quad \dfrac{}{t^{T} \in T}}{\mathit{false}}}{t \in [T \mid \neg P]} \quad \dfrac{\dfrac{}{\neg t.P}\,(1) \quad \dfrac{}{t^{T} \in T}}{t \in [T \mid \neg P]}\,(S^{+})}{t \in [T \mid \neg P]}\,(1)$$

and:

$$\dfrac{\dfrac{\dfrac{\dfrac{t \in [T \mid \neg P]}{\neg t.P}\,(S_{1}^{-}) \quad \dfrac{\overline{t \in [T \mid P]}}{t.P}\,(S_{1}^{-})}{\mathit{false}}}{t \notin [T \mid P]}}{t \in \neg[T \mid P]}\,(S_{\neg}^{+})$$

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:

$$\dfrac{\mathit{Pre}\ (U_0 \wedge U_1)\ t}{\mathit{Pre}\ U_i\ t}\ (\mathit{Pre}_{\wedge_i}^{-}).$$

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:

$$\frac{\mathit{Pre}\ U_i\ t}{\mathit{Pre}\ (U_0 \vee U_1)\ t}\ (\mathit{Pre}^{+}_{\vee_i})$$

$$\frac{\mathit{Pre}\ (U_0 \vee U_1)\ t \quad \mathit{Pre}\ U_0\ t \vdash P \quad \mathit{Pre}\ U_1\ t \vdash P}{P}\ (\mathit{Pre}^{-}_{\vee})$$

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

$$\mathit{Pre}\ (U_0 \vee U_1)\ t \Leftrightarrow \mathit{Pre}\ U_0\ t \vee \mathit{Pre}\ U_1\ t$$

5.2.3 The Precondition for Composition

We will deal with instances of composition where the operation schema expression $U_0 \fatsemi U_1$ has the type $\mathbb{P}(T_0 \curlyvee T_1')$ and where U_0 is of type $\mathbb{P}(T_0 \curlyvee T_2')$ and U_1 is of type $\mathbb{P}(T_2 \curlyvee T_1')$.

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

$$\frac{t_0 \star t_1' \in U_0 \quad \mathit{Pre}\ U_1\ t_1}{\mathit{Pre}\ (U_0 \fatsemi U_1)\ t_0}\ (\mathit{Pre}^{+}_{\fatsemi})$$

$$\frac{\mathit{Pre}\ (U_0 \fatsemi U_1)\ t_0 \quad \mathit{Pre}\ U_1\ y, t_0 \star y' \in U_0 \vdash P}{P}\ (\mathit{Pre}^{-}_{\fatsemi}).$$

The usual side-conditions apply to the eigenvariable y.

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

$$\frac{\mathit{Pre}\ (U_0 \fatsemi U_1)\ t_0}{\mathit{Pre}\ U_0\ t_0}$$

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:

$$\frac{\mathit{Pre}\ U\ t}{\mathit{Pre}\ (\exists\, \mathsf{z}, \mathsf{z}' : T^{\mathsf{z}} \bullet U)\ t}\ (\mathit{Pre}^{+}_{\exists})$$

$$\frac{\mathit{Pre}\ (\exists\, \mathsf{z}, \mathsf{z}' : T^{\mathsf{z}} \bullet U)\ t \quad \mathit{Pre}\ U\ y, y \doteq t \vdash P}{P}\ (\mathit{Pre}^{-}_{\exists}).$$

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 $\perp$. Such a value must be separated from the interpretation of Z and this is easily achieved by introducing a simple $\mathcal{Z_C}$ theory which we call $\mathcal{Z}_{\mathcal{C}}^{\perp}$.

In $\mathcal{Z}_{\mathcal{C}}^{\perp}$ we introduce new constants ("abortive" values), postulating new constants $\perp^T$ for every type T: these are usually called "lifted" types. There are, additionally, a number of axioms which ensure that all the new $\perp^T$ values interact properly.

$$\frac{}{\langle\!|\ \mathtt{z}_0 \Rrightarrow \perp^{T_0} \cdots \mathtt{z}_n \Rrightarrow \perp^{T_n} |\!\rangle = \perp^{[\mathtt{z}_0^{\mathtt{T}_0} \cdots \mathtt{z}_\mathtt{n}^{\mathtt{T}_\mathtt{n}}]}}$$

$$\frac{}{(\perp^{T_0}, \perp^{T_1}) = \perp^{T_0 \times T_1}}$$

$$\frac{}{\{z^T \mid z = \perp^T\} = \perp^{\mathbb{P}\, T}}$$

For example:

$$\perp^{[\mathtt{z}_0^{\mathtt{T}_0} \cdots \mathtt{z}_\mathtt{n}^{\mathtt{T}_\mathtt{n}}]}.\mathtt{z}_i = \perp^{T_i} \qquad (0 \leq i \leq n).$$

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

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

$$\Upsilon =_{df} \{z^{\Upsilon} \mid z \neq \perp\}$$

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

Further details, including the fact that the theory $\mathcal{Z}_{\mathcal{C}}^{\perp}$ is conservative over $\mathcal{Z_C}$, can be found in [11].

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

$$T_{\perp} =_{df} T \cup \{\perp\}$$

and

$$T^{\star} =_{df} T^{in}_{\perp} \star T^{out'}_{\perp};$$

then:

$$\overset{\bullet}{U} =_{df} \{z_0 \star z_1' \in T^{\star} \mid Pre\ U\ z_0 \Rightarrow z_0 \star z_1' \in U\}$$

which leads to rules for lifted totalised sets:

$$\frac{t_0 \star t_1' \in T^{\star} \quad Pre\ U\ t_0 \ \vdash\ t_0 \star t_1' \in U}{t_0 \star t_1' \in \overset{\bullet}{U}}\ (\bullet^{+})$$

and

$$\frac{t_0 \star t_1' \in \overset{\bullet}{U} \quad Pre\ U\ t_0}{t_0 \star t_1' \in U}\ (\bullet^{-}_{\text{o}}) \qquad \frac{t_0 \star t_1' \in \overset{\bullet}{U}}{t_0 \star t_1' \in T^{\star}}\ (\bullet^{-}_{1}).$$

The following are also derivable:

$$\frac{}{U \subseteq \overset{\bullet}{U}}\ (i) \qquad \frac{}{\perp \in \overset{\bullet}{U}}\ (ii) \qquad \frac{\neg Pre\ U\ t_0 \quad t_0 \in T^{in}_{\perp} \quad t_1' \in T^{out'}_{\perp}}{t_0 \star t_1' \in \overset{\bullet}{U}}\ (iii).$$

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.

$W_{\bullet}$-refinement, written $U_0 \sqsupseteq_{w_{\bullet}} U_1$ is defined by:

$$U_0 \sqsupseteq_{w_{\bullet}} U_1 =_{df} \overset{\bullet}{U_0} \subseteq \overset{\bullet}{U_1}$$

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:

$$\frac{Pre\ U_1\ z\ \vdash\ Pre\ U_0\ z \quad Pre\ U_1\ z_0, z_0 \star z_1' \in U_0\ \vdash\ z_0 \star z_1' \in U_1}{U_0 \sqsupseteq_s U_1}\ (\sqsupseteq^{+}_{s})$$

$$\frac{U_0 \sqsupseteq_s U_1 \quad Pre\ U_1\ t}{Pre\ U_0\ t}\ (\sqsupseteq^{-}_{s_0})$$

$$\frac{U_0 \sqsupseteq_s U_1 \quad Pre\ U_1\ t_0 \quad t_0 \star t_1' \in U_0}{t_0 \star t_1' \in U_1}\ (\sqsupseteq^{-}_{s_1})$$

The theories $\sqsupseteq_{w_\bullet}$ 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 $\mathcal{Z}_{\mathcal{C}}$.

First we have post-sets:

$$Post\ \ U\ \ z_0 =_{df} \{z_1' \mid z_0 \star z_1' \in U\}$$

This permits a wp-interpretation for schemas

$$wp\ \ U\ \ C =_{df} \{z \mid Pre\ \ U\ \ z \wedge Post\ \ U\ \ z \subseteq C\}$$

leading to the following rules:

$$\frac{Pre\ \ U\ \ t \quad z' \in Post\ \ U\ \ t\ \vdash\ z' \in C}{t \in wp\ \ U\ \ C}$$

where z is a fresh variable.

$$\frac{t \in wp\ \ U\ \ C}{Pre\ \ U\ \ t} \qquad \frac{t_0 \in wp\ \ U\ \ C \quad t_1' \in Post\ \ U\ \ t_0}{t_1' \in C}\ .$$

We can now define WP-refinement:

$$U_0 \sqsupseteq_{wp} U_1 =_{df} \forall\, C^{\mathbb{P}\, T^{out'}} \bullet wp\ \ U_1\ \ C \subseteq wp\ \ U_0\ \ C$$

leading to the following introduction and elimination rules

$$\frac{z \in wp\ \ U_1\ \ C\ \vdash\ z \in wp\ \ U_0\ \ C}{U_0 \sqsupseteq_{wp} U_1}\ (\sqsupseteq^{+}_{wp})$$

where z and C are fresh variables:

$$\frac{U_0 \sqsupseteq_{wp} U_1 \quad t \in wp\ \ U_1\ \ C}{t \in wp\ \ U_0\ \ C}\ (\sqsupseteq^{-}_{wp}).$$

The theory $\sqsupseteq_{wp}$ is also equivalent to $\sqsupseteq_{w_\bullet}$ 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 $\mathcal{Z}_{\mathcal{C}}$ is straightforward.

We begin with the lifting of simulations:

$$S^{\mathbb{P}(\overset{\circ}{T_1} \curlyvee T_0')} =_{df} \{z_1 \star z_0' \in T_{1_\perp} \star T_{0_\perp}' \mid z_1 \neq \perp \Rightarrow z_1 \star z_0' \in S\}$$

leading to the following rules:

$$\frac{t_1 \star t'_0 \in T_{1_\perp} \star T'_{0_\perp} \quad t_1 \neq \perp \vdash t_1 \star t'_0 \in S}{t_1 \star t'_0 \in \mathring{S}}\ (\circ^+) \qquad \frac{t_1 \star t'_0 \in \mathring{S} \quad t_1 \neq \perp}{t_1 \star t'_0 \in S}\ (\circ^-_0)$$

$$\frac{t_1 \star t'_0 \in \mathring{S}}{t_1 \star t'_0 \in T_{1_\perp} \star T'_{0_\perp}}\ (\circ^-_1).$$

Then we can define WF$_\bullet$-refinement, a theory of forward simulation data refinement:

$$U_0 \sqsupseteq_{wf_\bullet} U_1 =_{df} \mathring{S} \fatsemi \overset{\bullet}{U_0} \subseteq \overset{\bullet}{U_1} \fatsemi \mathring{S}$$

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

$$\frac{z_1 \star z'_0 \in \mathring{S} \fatsemi \overset{\bullet}{U_0} \vdash z_1 \star z'_0 \in \overset{\bullet}{U_1} \fatsemi \mathring{S}}{U_0 \sqsupseteq_{wf_\bullet} U_1}\ (\sqsupseteq^+_{wf_\bullet}) \qquad \frac{U_0 \sqsupseteq_{wf_\bullet} U_1 \quad t_1 \star t'_0 \in \mathring{S} \fatsemi \overset{\bullet}{U_0}}{t_1 \star t'_0 \in \overset{\bullet}{U_1} \fatsemi \mathring{S}}\ (\sqsupseteq^-_{wf_\bullet}).$$

As with operation refinement, it is possible to define an equivalent theory (SF-refinement) based solely on the language. Let x_0, x_1, z_0, z_1, z_2 be fresh variables:

$$\frac{\begin{array}{rcl} z_1 \star z'_0 \in S, \mathit{Pre}\ U_1\ z_1 & \vdash & \mathit{Pre}\ U_0\ z_0 \\ \mathit{Pre}\ U_1\ x_1, x_0 \star z'_2 \in U_0, x_1 \star x'_0 \in S & \vdash & x_1 \star t' \in U_1 \\ \mathit{Pre}\ U_1\ x_1, x_0 \star z'_2 \in U_0, x_1 \star x'_0 \in S & \vdash & t \star z'_2 \in S \end{array}}{U_0 \sqsupseteq_{sf} U_1}\ (\sqsupseteq^+_{sf})$$

$$\frac{U_0 \sqsupseteq_{sf} U_1 \quad \mathit{Pre}\ U_1\ t_1 \quad t_1 \star t'_0 \in S}{\mathit{Pre}\ U_0\ t_0}\ (\sqsupseteq^-_{sf_0})$$

$$\frac{U_0 \sqsupseteq_{sf} U_1 \quad \mathit{Pre}\ U_1\ t_1 \quad t_0 \star t'_2 \in U_0 \quad t_1 \star t'_0 \in S \quad t_1 \star y' \in U_1, y \star t'_2 \in S \vdash P}{P}\ (\sqsupseteq^-_{sf_1}).$$

The usual side-conditions apply to the eigenvariable y.

The theories $\sqsupseteq_{sf}$ and $\sqsupseteq_{wf_\bullet}$ 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:

$$\frac{\begin{array}{rcl} x \star z' \in S \Rightarrow \mathit{Pre}\ U_1\ z & \vdash & \mathit{Pre}\ U_0\ x \\ z_0 \star z' \in S \Rightarrow \mathit{Pre}\ U_1\ z, x_0 \star x'_1 \in S, z_0 \star x'_0 \in U_0 & \vdash & z_0 \star t' \in S \\ z_0 \star z' \in S \Rightarrow \mathit{Pre}\ U_1\ z, x_0 \star x'_1 \in S, z_0 \star x'_0 \in U_0 & \vdash & t \star x'_1 \in U_1 \end{array}}{U_0 \sqsupseteq_{sb} U_1}\ (\sqsupseteq^+_{sb})$$

$$\frac{U_0 \sqsupseteq_{sb} U_1 \quad t \star z' \in S \vdash \mathit{Pre}\ U_1\ z}{\mathit{Pre}\ U_0\ t}\ (\sqsupseteq^-_{sb_0})$$

$$\frac{U_0 \sqsupseteq_{sb} U_1 \qquad t_1 \star t'_2 \in S \quad \begin{array}{c} t_0 \star z' \in S \vdash \mathit{Pre}\ U_1\ z \\ t_0 \star t'_1 \in U_0 \end{array} \quad t_0 \star y' \in S, y \star t'_2 \in U_1 \vdash P}{P}\ (\sqsupseteq^-_{sb_1}).$$

The usual side-conditions apply to the eigenvariable y.

WB$_\bullet$-refinement is: Let z_0, z_1 be fresh.

$$\frac{z_0 \star z'_1 \in \overset{\bullet}{U}_0 \fatsemi \overset{\circ}{S} \vdash z_0 \star z'_1 \in \overset{\circ}{S} \fatsemi \overset{\bullet}{U}_1}{U_0 \overset{s}{\sqsupseteq}_{wb_\bullet} U_1}\ (\sqsupseteq^+_{wb_\bullet}) \qquad \frac{U_0 \overset{s}{\sqsupseteq}_{wb_\bullet} U_1 \quad t_0 \star t'_1 \in \overset{\bullet}{U}_0 \fatsemi \overset{\circ}{S}}{t_0 \star t'_1 \in \overset{\circ}{S} \fatsemi \overset{\bullet}{U}_1}\ (\sqsupseteq^-_{wb_\bullet})$$

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:

$$[C^{\mathbb{P}\,T_1}]S^{\mathbb{P}(T_1 \curlyvee T'_0)} =_{df} \{z_0 \in T_0 \mid \exists\, z_1 \in C \bullet z_1 \star z'_0 \in S\}.$$

This leads to the following theory, WPF-refinement:

$$\frac{z \in [\mathit{wp}\ U_1\ C]S \vdash z \in \mathit{wp}\ U_0\,[C]S'}{U_0 \sqsupseteq_{wpf} U_1}\ (\sqsupseteq^+_{wpf})$$

where z and C are fresh variables.

$$\frac{U_0 \sqsupseteq_{wpf} U_1 \quad t \in [\mathit{wp}\ U_1\ C]S}{t \in \mathit{wp}\ U_0\,[C]S'}\ (\sqsupseteq^-_{wpf})$$

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.

$$S^{\mathbb{P}(T_0 \curlyvee T'_1)}[C^{\mathbb{P}\,T_1}] =_{df} \{z_0 \in T_0 \mid \forall\, z_1 \bullet z_0 \star z'_1 \in S \Rightarrow z_1 \in C\}$$

This leads to WPB-refinement:

$$\frac{z \in S[\mathit{wp}\ U_1\ C] \vdash z \in \mathit{wp}\ U_0\ S'[C]}{U_0 \sqsupseteq_{wpb} U_1}\ (\sqsupseteq^+_{wpb})$$

where z and C are fresh variables.

$$\frac{U_0 \sqsupseteq_{wpb} U_1 \quad t \in S[\mathit{wp}\ U_1\ C]}{t \in \mathit{wp}\ U_0\ S'[C]}\ (\sqsupseteq^-_{wpb}).$$

The theories $\sqsupseteq_{wpf}$ and $\sqsupseteq_{wpb}$ are equivalent to (and undoubtedly simpler than) $\sqsupseteq_{sf}$ and $\sqsupseteq_{sb}$ (hence to $\sqsupseteq_{wf_\bullet}$ and $\overset{s}{\sqsupseteq}_{wb_\bullet}$) 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

$$\forall\, z \bullet Pre\ U_0\ z \wedge Pre\ U_2\ z \Rightarrow Pre\,(U_0 \wedge U_2)\, z$$

then we also have

$$\frac{U_0 \sqsupseteq_s U_1}{U_0 \wedge U_2 \sqsupseteq_s U_1 \wedge U_2}\ .$$

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:

$$\forall\, z \bullet Pre\ U_0\ z \wedge Pre\ U_2\ z \Rightarrow Pre\ U_1\ z.$$

Then the following rule is derivable:

$$\frac{U_0 \sqsupseteq_s U_1}{U_0 \vee U_2 \sqsupseteq_s U_1 \vee U_2}$$

In this case let us consider the $\mathcal{Z}_\mathcal{C}$ proof:

$$\dfrac{\dfrac{\dfrac{}{Pre\,(U_1 \vee U_2)\,z}\,(1) \quad \dfrac{\dfrac{U_0 \sqsupseteq_s U_1 \quad \dfrac{}{Pre\ U_1\ z}\,(2)}{Pre\ U_0\ z}\,(\sqsupseteq_{s_0}^{-})}{Pre\,(U_0 \vee U_2)\,z}\,(Pre^{+}_{\vee_0}) \quad \dfrac{\dfrac{}{Pre\ U_2\ z}\,(2)}{Pre\,(U_0 \vee U_2)\,z}\,(Pre^{+}_{\vee_1})}{Pre\,(U_0 \vee U_2)\,z}\,(Pre^{-}_{\vee}, 2) \quad \begin{array}{c}\delta_0\\ \vdots\end{array}}{U_0 \vee U_2 \sqsupseteq_s U_1 \vee U_2}\,(\sqsupseteq_s^{+}, 1)$$

where δ_0 stands for the following branch (where we write z for $z_0 \star z_1'$)

$$\dfrac{\dfrac{}{z \in U_0 \vee U_2}\,(1) \qquad \dfrac{\dfrac{U_0 \sqsupseteq_s U_1 \quad \begin{matrix}\delta_1\\ \vdots\\ \mathit{Pre}\; U_1\; z_0\end{matrix} \quad \dfrac{}{z \mathrel{\dot{\in}} U_0}\,(3)}{z \mathrel{\dot{\in}} U_1}\,(\sqsupseteq^-_{S_1})}{z \in U_1 \vee U_2}\,(S^+_{\vee_0}) \qquad \dfrac{\dfrac{}{z \mathrel{\dot{\in}} U_2}\,(3)}{z \in U_1 \vee U_2}\,(S^+_{\vee_1})}{z \in U_1 \vee U_2}\,(S^-_{\vee}, 3)$$

and δ_1 is

$$\dfrac{\dfrac{}{\mathit{Pre}\;(U_1 \vee U_2)\; z_0}\,(1) \qquad \dfrac{}{\mathit{Pre}\; U_1\; z_0}\,(4) \qquad \begin{matrix}\dfrac{\dfrac{\dfrac{}{z_0 \star z_1' \mathrel{\dot{\in}} U_0}\,(3)}{\mathit{Pre}\; U_0\; z_0} \quad \dfrac{}{\mathit{Pre}\; U_2\; z_0}\,(4)}{\mathit{Pre}\; U_0\; z_0 \wedge \mathit{Pre}\; U_2\; z_0}\\ \vdots\\ \mathit{Pre}\; U_1\; z_0\end{matrix}}{\mathit{Pre}\; U_1\; z_0}\,(\mathit{Pre}^-_{\vee}, 4)$$

Note the point (in the right-most sub-proof of δ_1) where the side-condition is required. A proof attempt without the sidecondition 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 $\mathcal{Z_C}$, 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 θ-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 μ-charts [14]. Once the definitions that formalise them have been made, and the Z logic rules are expressed via the definitions as μ-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 μ-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

[1] Abrial, J. R.: The B-Book. Cambridge University Press, 1996.

[2] Arthan,R. D.: On free type definitions in Z. In [27], pp. 40–58, 1992.

[3] Barden, R.—Stepney, S.—Cooper, D.: Z in Practice. Prentice Hall, BCS Practitioner Series, 1994.

[4] Bowen, J. P.—Gordon, M. J. C.: A Shallow Embedding of Z in HOL. Information and Software Technology, Vol. 37, 1995, Nos. 5–6, pp. 269–276.

[5] Brien, S.: A Model and Logic for Generically Typed Set Theory Z. (draft) D. Phil. thesis, University of Oxford, 1995.

[6] Brien, S.: A Logic and Model for the Z Standard. D. Phil. thesis, Oxford University Computing Laboratory, 1999.

[7] Brien, S.—Martin, A.: A Tutorial of Proof in Standard Z. Technical Monograph PRG-120, Oxford University Computing Laboratory, 1996.

[8] Deutsch, M.—Henson, M. C.: An Analysis of Backward Simulation Data Refinement. Proc. Refinement for Critical Systems RCS '03. University of Essex, Department of Computer Science Technical Report CSM-383, 2003.

[9] Deutsch, M.—Henson, M. C.: An Analysis of Forward Simulation Data Refinement. In D. Bert, J. P. Bowen, S. King, and M. Waldén, editors, ZB 2003: Formal Specification and Development in Z and B, Volume 2651 of Lecture Notes in Computer Science, pp. 148–167, Springer-Verlag, June 2003.

[10] Deutsch, M.—Henson, M. C.: An Analysis of Total Correctness Refinement Models for Partial Relation Semantics II. Logic Journal of the IGPL, Vol. 11, No. 3, pp. 319–352.

[11] Deutsch, M.—Henson, M. C.—Reeves, S.: An Analysis of Total Correctness Refinement Models for Partial Relation Semantics I. Logic Journal of the IGPL, Vol. 11, No. 3, pp. 287–317.

[12] Deutsch, M.—Henson, M. C.—Reeves, S.: Operation Refinement and Monotonicity in the Schema Calculus. In D. Bert, J. P. Bowen, S. King, and M. Waldén, editors, ZB 2003: Formal Specification and Development in Z and B, Volume 2651 of Lecture Notes in Computer Science, pp. 103–126. Springer-Verlag, Berlin, June 2003.

[13] Fergus, E.—Ince, D. C.: Z Specifications and Modal Logic. In P. A. V. Hall, editor, Proc. Software Engineering 90, Volume 1 of British Computer Society Conference Series. Cambridge University Press, 1990.

[14] Goldson, D.—Reeve, G.—Reeves, S.: μ-chart-based Specification and Refinement. In C. George and H. Miao, editors, Formal Methods and Software Engineering, Volume 2495 of Lecture Notes in Computer Science, pp. 323–334. Springer-Verlag, Berlin, October 2002.

[15] Hall, J.—Martin, A.: $\mathcal{W}$ Reconstructed. In J. P. Bowen, M. G. Hinchey, and D. Till, editors, ZUM'97: The Z Formal Specification Notation, Volume 1212 of Lecture Notes in Computer Science, pp. 116–134. Springer-Verlag, 1997.

[16] Henson, M. C.: The Standard Logic of Z is Inconsistent. Formal Aspects of Computing, Vol. 10, 1998, No. 3, pp. 243–247.

[17] Henson, M. C.—Reeves, S.: Revising Z: I – Logic and Semantics. Formal Aspects of Computing, Vol. 11, 1999, No. 4, pp. 359–380.

[18] Henson, M. C.—Reeves, S.: Revising Z: II – Logical Development. Formal Aspects of Computing, Vol. 11, 1999, No. 4, pp. 381–401.

[19] Henson, M. C.—Reeves, S.: Investigating Z. Journal of Logic and Computation, Vol. 10, 2000, No. 1, pp. 43–73.

[20] Henson, M. C.—Reeves, S.: Program Development and Specification Refinement in the Schema Calculus. In J. P. Bowen, S. Dunne, A. Galloway, and S. King, editors, ZB 2000: Formal Specification and Development in Z and B, Volume 1878 of Lecture Notes in Computer Science, pp. 344–362. Springer-Verlag, Berlin, 2000.

[21] Henson, M. C.—Reeves, S.: A Logic for Schema-Based Program Development. Formal Aspects of Computing, Vol. 15, 2003, No. 1, pp. 48–83.

[22] King, S.: The Standard Logic for Z: A clarification. Formal Aspects of Computing Journal, Vol. 11, 1999, No. 4, pp. 472–473.

[23] Martin, A.: Encoding $\mathcal{W}$: A logic for Z in 2OBJ. In Woodcock and Larsen [34], pp. 462–481.

[24] Martin, A.: A Revised Deductive System for Z. Technical Report TR98-21, SVRC, University of Queensland, 1998.

[25] Morgan, C. C.: Programming from Specifications. Prentice Hall International Series in Computer Science, 2nd edition, 1994.

[26] Nicholls, J. editor: ISO Committee Draft: Z Notation, Version 1.2. Z Standards Panel, 1995.

[27] Nicholls, J. E. editor: Z User Workshop, York 1991, Workshops in Computing. Springer-Verlag, London, 1992.

[28] Spivey, J. M.: Understanding Z: A Specification Language and its Formal Semantics. Cambridge University Press, 1988.

[29] Spivey, J. M.: The Z Notation: A Reference Manual. Prentice Hall International Series in Computer Science, 2nd edition, 1992.

[30] Spivey, J. M.: The Consistency Theorem for Free Type Definitions in Z. Formal Aspects of Computing Journal, Vol. 8, 1996, No. 3, pp. 369–376.

[31] Woodcock, J. C. P.: Calculating Properties of Z Specifications. ACM SIGSOFT Software Engineering Notes, Vol. 14, 1989, No. 5, pp. 43–54.

[32] Woodcock, J. C. P.—Brien, S.: $\mathcal{W}$: A Logic for Z. In [27], pp. 77–96, 1992.

[33] Woodcock, J. C. P.—Davies, J.: Using Z: Specification, Refinement and Proof. Prentice Hall International Series in Computer Science, 1996.

[34] Woodcock, J. C. P.—Larsen, P. G. editors: FME'93: Industrial-Strength Formal Methods, Volume 670 of Lecture Notes in Computer Science. Formal Methods Europe, Springer-Verlag, Berlin, 1993.

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 constructivize 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.