Formal Aspects of Computing (1999) 11: 381–401


**Formal Aspects of Computing**

# Revising Z: Part II – logical development

Martin C. Henson[1] and Steve Reeves[2]

[1]Department of Computer Science, University of Essex, UK;
[2]Department of Computer Science, University of Waikato, New Zealand

**Keywords:** Specification language Z; Logics of specification languages; Semantics of specification languages

**Abstract.** This is the second of two related papers. In "Revising Z: Part I - logic and semantics" (this journal) we introduced a simple specification logic $Z_C$ comprising a logic and a semantics (in ZF set theory). We then provided an interpretation for (a rational reconstruction of) the specification language Z within $Z_C$. As a result we obtained a sound logic for Z, including the basic schema calculus. In this paper we extend the basic framework with more sophisticated features (including schema operations) and we mount a critique of a number of concepts used in Z. We further demonstrate that the complications and confusions which these concepts introduce can be avoided without compromising expressibility.

## 1. Introduction

In the earlier companion paper [HeR99] we introduced a specification system $Z_C$, a typed set theory incorporating the notion of a *schema type* and we established a number of meta-mathematical results including its soundness with respect to a simple set-theoretic semantics. We went on to introduce a notation very much closer to the Z familiar in the literature. This included more substantial mechanisms for the construction of propositions, sets and, in particular, schemas.

In this paper we further extend the logic for Z substantially. In Section 2, we extend the schema calculus with a number of new forms of expression and introduce a number of other new forms of term, set and propositional forms. We then turn, in Section 3, to consider two Z concepts, $\theta$ expressions and schema priming, which are without doubt the locus of much confusion in the literature.

*Correspondence and offprint requests to*: Steve Reeves, Department of Computer Science, University of Waikato, Private Bag 3105, Hamilton, New Zealand

After reviewing these notions we go on to provide additional mechanisms suitable to either interpret or modify them. Finally, in Section 4, we examine some example specifications from the literature which employ the full range of the apparatus available in Z and demonstrate how such specifications may be rendered in our framework.

We do not review the logical development of [HeR99] in this paper, and the reader will need to refer to that in order to establish the logical systems which are the point of departure here.

## 2. Derived constructs

We indicated in [HeR99] that, given the basic connectives and set (hence schema) operations we introduced, we would be able to construct others that we expect to find in Z. Since we are particularly concerned with developing the logic of Z, it is appropriate that we go on to examine the expected logical consequences of these constructions. We shall, in particular, spend some time extending the schema calculus which is of major importance in Z. In fact it is not quite clear why this has traditionally been referred to as a schema *calculus* since the literature typically introduces nothing beyond a *notation* for schema expressions. It is true that there are some suggested rules for membership in compound schemas, at least for the simplest cases like conjunction, in the normative source [Nic95], but these are quite clearly wrong. For example (*ibid*. Section F.6.6, p. 207) we can easily derive the following rules, the first of which is too restrictive and the others are not even well-formed:

$$\frac{\Gamma \vdash b \in S \quad \Gamma \vdash b \in T}{\Gamma \vdash b \in S \land T} \qquad \frac{\Gamma \vdash b \in S \land T}{\Gamma \vdash b \in S} \qquad \frac{\Gamma \vdash b \in S \land T}{\Gamma \vdash b \in T}$$

using the rules (*SchBindMem*) and (*SAnd*). We are not the only authors to detect this mistake in [Nic95]. However, the suggested remedy (adding a proviso to rule (*SchBindMem*) [Mar98]) only ensures that the elimination rules above are well-defined: they are still very limited. Beyond this incorrect treatment of the simplest aspects of the schema calculus there is almost nothing, for example the incomplete rule for schema composition (*SComp*) (*ibid*. p. 208). These errors were carried over into [BrM96] but are rectified in [Mar98].

### 2.1. Schema conjunction

We would expect to define conjunction over schemas by analogy with operations like set intersection and logical conjunction:

$$[\![S_0 \land S_1]\!] =_{df} [\![\neg(\neg S_0 \lor \neg S_1)]\!]$$

Using rules $(Z_{\neg S})$ and $(Z_{\lor S})$[1] we obtain the following derived rule for type assignment:

$$\frac{\Gamma \rhd S_0 : \mathbb{P}\, T_0 \quad \Gamma \rhd S_1 : \mathbb{P}\, T_1}{\Gamma \rhd S_0 \land S_1 : \mathbb{P}(T_0 \lor T_1)}\ (Z_{\land S})$$

[1] *A reminder*: these rules and many others that follow are from [HeR99].

We can simplify the right-hand side of the definition, so that we have, for fresh $y$:

$$\left[\!\left[ S_0^{\mathbb{P}\, T_0} \wedge S_1^{\mathbb{P}\, T_1} \right]\!\right] = \{ y : T_0 \vee T_1 \mid y \upharpoonright T_0 \in [\![S_0]\!] \wedge y \upharpoonright T_1 \in [\![S_1]\!] \}$$

The following introduction rule is derivable: $(\wedge^+)$ and $(\{\}^+)$:

$$\frac{\Gamma \vdash t \upharpoonright T_0 \in S_0 \quad \Gamma \vdash t \upharpoonright T_1 \in S_1 \quad \Gamma^- \rhd t : T_0 \vee T_1}{\Gamma \vdash t \in S_0 \wedge S_1} \; (S^+_\wedge)$$

The corresponding elimination rules:

$$\frac{\Gamma \vdash t \in S_0 \wedge S_1 \quad \Gamma^- \rhd S_0 : \mathbb{P}\, T_0}{\Gamma \vdash t \upharpoonright T_0 \in S_0} \; (S^-_{\wedge_0})$$

$$\frac{\Gamma \vdash t \in S_0 \wedge S_1 \quad \Gamma^- \rhd S_1 : \mathbb{P}\, T_1}{\Gamma \vdash t \upharpoonright T_1 \in S_1} \; (S^-_{\wedge_1})$$

With these in place we can prove the expected relationship (see [WoD96] pp. 165-6):

**Lemma 2.1.**

$$\frac{\Gamma^- \rhd [D_0^* \mid P_0] : \mathbb{P}\, T_0 \quad \Gamma^- \rhd [D_1^* \mid P_1] : \mathbb{P}\, T_1}{\Gamma \vdash [D_0^* \mid P_0] \wedge [D_1^* \mid P_1] = [D_0^* \vee D_1^* \mid P_0 \wedge P_1]} \; (\wedge^=)$$

*Proof.* We can use the equational logic that we already have at our disposal. The result follows easily by rules $(\neg^=)$ and $(\vee^=)$ and De Morgan's laws.

Finally we have the substitution rule:

$$\frac{S_0 = S_2 \quad S_1 = S_2}{S_0 \wedge S_1 = S_2 \wedge S_3}$$

Again, these follow easily from the corresponding rules for disjunction and negation schemas.

## 2.2. Schema implication

Following the pattern given above for conjunction, we would expect the definition:

$$[\![S_0 \Rightarrow S_1]\!] =_{df} [\![\neg S_0 \vee S_1]\!]$$

Using the rules $(Z_{\neg S})$ and $(Z_{\vee S})$ we obtain a derived rule for type assignment:

$$\frac{\Gamma \rhd S_0 : \mathbb{P}\, T_0 \quad \Gamma \rhd S_1 : \mathbb{P}\, T_1}{\Gamma \rhd S_0 \Rightarrow S_1 : \mathbb{P}(T_0 \vee T_1)} \; (Z_{\Rightarrow S})$$

Simplifying the right-hand side of the definition, we obtain, for fresh $z$:

$$\left[\!\left[ S_0^{\,T_0} \Rightarrow S_1^{\,T_1} \right]\!\right] = \{ z : T_0 \vee T_1 \mid z \upharpoonright T_0 \notin [\![S_0]\!] \vee z \upharpoonright T_1 \in [\![S_1]\!] \}$$

The introduction rule is:

$$\frac{\Gamma, t \upharpoonright T_0 \in S_0 \vdash t \upharpoonright T_1 \in S_1 \quad \Gamma^- \rhd t : T_0 \vee T_1 \quad \Gamma^- \rhd S_0 : \mathbb{P}\, T_0}{\Gamma \vdash t \in S_0 \Rightarrow S_1} \; (S^+_\Rightarrow)$$

The elimination rule is:

$$\frac{\Gamma \vdash t \in S_0 \Rightarrow S_1 \quad \Gamma \vdash t \upharpoonright T_0 \in S_0 \quad \Gamma^- \triangleright S_1 : \mathbb{P}\, T_1}{\Gamma \vdash t \upharpoonright T_1 \in S_1} \; (S^-_{\Rightarrow})$$

The expected relationship holds:

**Lemma 2.2.**

$$\frac{\Gamma^- \triangleright [D_0^* \mid P_0] : \mathbb{P}\, T_0 \quad \Gamma^- \triangleright [D_1^* \mid P_1] : \mathbb{P}\, T_1}{\Gamma \vdash [D_0^* \mid P_0] \Rightarrow [D_1^* \mid P_1] = [D_0^* \vee D_1^* \mid P_0 \Rightarrow P_1]}$$

*Proof.* Using the equational logic: rules $(\neg^=)$ and $(\vee^=)$. □

Finally we have the expected substitution rule:

$$\frac{S_0 = S_1 \quad S_2 = S_3}{S_0 \Rightarrow S_2 = S_1 \Rightarrow S_3}$$

## 2.3. Schema inclusion

Schema inclusion can be defined in terms of schema conjunction.

$$[\![[D_0;\ [D_1 \mid P_1] \mid P_0]]\!] =_{df} [\![[D_0 \vee D_1 \mid P_0] \wedge [D_1 \mid P_1]]\!]$$

It should be noted that this, unlike the other operators we consider, is a non-compositional definition which involves a generalisation, on the left-hand side, of the language of declarations to include schema references.

The rules are then easily calculated as special cases of those for schema conjunction. First the typing rules:

$$\frac{\Gamma \triangleright [D_0 \vee D_1 \mid P_0] : \mathbb{P}(T_0 \vee T_1) \quad \Gamma \triangleright [D_1 \mid P_1] : \mathbb{P}\, T_1}{\Gamma \triangleright [D_0;\ [D_1 \mid P_1] \mid P_1] : \mathbb{P}(T_0 \vee T_1)} \; (Z_{inc})$$

The introduction rule is:

$$\frac{\Gamma \vdash t \in [D_0 \vee D_1 \mid P_0] \quad \Gamma \vdash t \upharpoonright T_1 \in [D_1 \mid P_1]}{\Gamma \vdash t \in [D_0;\ [D_1 \mid P_1] \mid P_0]}$$

The elimination rules are:

$$\frac{\Gamma \vdash t \in [D_0;\ [D_1 \mid P_1] \mid P_0]}{\Gamma \vdash t \in [D_0 \vee D_1 \mid P_0]}$$

$$\frac{\Gamma^- \triangleright [D_1 \mid P_1] : \mathbb{P}\, T_1 \quad \Gamma \vdash t \in [D_0;\ [D_1 \mid P_1] \mid P_0]}{\Gamma \vdash t \upharpoonright T_1 \in [D_1 \mid P_1]}$$

Finally, we have the expected equational law:

$$\frac{\Gamma^- \triangleright [D_0^* \mid P_0] : \mathbb{P}\, T_0 \quad \Gamma^- \triangleright [D_1^* \mid P_1] : \mathbb{P}\, T_1}{\Gamma \vdash [D_0^*;\ [D_1^* \mid P_1] \mid P_0] = [D_0^* \vee D_1^* \mid P_0 \wedge P_1]} \; (inc^=)$$

Although the $\Delta$ and $\Xi$ schemas of Z are intimately linked, in the standard accounts, with schema inclusion, we shall delay their consideration until we discuss schema priming and the $\theta$ operator (Section 3 below).

## 2.4. Schema composition and piping

We proceed by adopting a standard definition of schema composition, in terms of renaming, conjunction and hiding (see *e.g.* [Dil94]). We give a very simplified, introductory account, restricting the composition along a single pair of complementary labels.

We shall need to index the composition operator with the pair of labels along which the composition is taken. Let $S_0 : \mathbb{P}(T_0 \vee [l' : T])$ and $S_1 : \mathbb{P}(T_1 \vee [l : T])$. Let $v$ be a fresh label. Then:

$$[\![ S_0 \fatsemi_{(l',l)} S_1 ]\!] =_{df} [\![ (S_0[l' \leftarrow v] \wedge S_1[l \leftarrow v]) \setminus [v : T] ]\!]$$

For notational simplicity it is sensible to define a derived operator on types:

$$T_0 \fatsemi_{(l',l)} T_1 =_{df} (T_0[l' \leftarrow v] \vee T_1[l \leftarrow v]) \setminus [v : T]$$

First we have the typing rule, which is easily derived:

$$\frac{S_0 : \mathbb{P}(T_0 \vee [l' : T]) \quad S_1 : \mathbb{P}(T_1 \vee [l : T])}{S_0 \fatsemi_{(l',l)} S_1 : \mathbb{P}(T_0 \fatsemi_{(l',l)} T_1)}$$

In the introduction rule we write $V_0$ and $V_1$ for $T_0 \vee [l' : T]$ and $T_1 \vee [l : T]$ respectively.

$$\frac{\Gamma \vdash (t \upharpoonright V_0[l' \leftarrow v])[v \leftarrow l'] \in S_0 \quad \Gamma \vdash (t \upharpoonright V_1[l \leftarrow v])[v \leftarrow l] \in S_1}{\Gamma \vdash t \upharpoonright (T_0 \fatsemi_{(l',l)} T_1) \in S_0^{\mathbb{P} V_0} \fatsemi_{(l',l)} S_1^{\mathbb{P} V_1}} \ (S_{\fatsemi}^{+})$$

This is calculated using the rules $(S_h^+)$, $(S_\wedge^+)$ and $(S_\leftarrow^+)$ twice.

We also obtain an elimination rule:

$$\frac{\Gamma \vdash t \in S_0^{\mathbb{P}(T_0 \vee [l' : T])} \fatsemi_{(l',l)} S_1^{\mathbb{P}(T_1 \vee [l : T])} \quad \Gamma' \vdash P}{\Gamma \vdash P} \ (S_{\fatsemi}^{-}), \text{ for fresh } y_0 \text{ and } y_1$$

where $\Gamma'$ is:
$\Gamma^-, y_0 : T_0 \vee [v : T], y_1 : T_1 \vee [v : T]$; $\Gamma^+, y_0[v \leftarrow l'] \in S_0, y_1[v \leftarrow l] \in S_1, y_0 \upharpoonright T_0 = t \upharpoonright T_0, y_1 \upharpoonright T_1 = t \upharpoonright T_1$.

The substitution rule is as expected:

$$\frac{\Gamma \vdash S_0^{\mathbb{P}(T_0 \vee [l' : T])} = S_2 \quad \Gamma \vdash S_1^{\mathbb{P}(T_1 \vee [l : T])} = S_3}{\Gamma \vdash S_0 \fatsemi_{(l',l)} S_1 = S_2 \fatsemi_{(l',l)} S_3}$$

Turning now to piping, we immediately see that the definition, and therefore the rules, are much the same. All we require is to select our complementary labels to be distinguished by the diacritical marks which indicate input and output. For example, the composition operator we have introduced above, indexed by a pair of labels $(l!, l?)$ implements the piping operator over a single input/output channel.

This section only begins an investigation of composition and piping and we have obtained rules which are rather unpleasant. Possibly the logic offers other methods for defining composition which would result in more manageable rules. Investigation of that is left to future work.

## 2.5. Schema restriction

In view of our filtering operation on terms which we have extended to sets, we can give a pleasant definition:

$$\left[\!\left[ S_0^{\mathbb{P}\, T_0} \upharpoonright S_1^{\mathbb{P}\, T_1} \right]\!\right] =_{df} [\![ S_0 \upharpoonright T_1 \wedge S_1 ]\!]$$

when $T_1 \preceq T_0$. When the schema $S_1$ is just a schema set we get:

$$S \upharpoonright [D^*] = S \upharpoonright [D]$$

This equation explains our use in this paper of the restriction symbol for term filtering. From filtered terms we were able to introduce filtered sets, and from those we obtain restricted schemas here. The overloading of the symbol is quite unambiguous: restriction involves a schema *set*, whereas filtering employs a schema *type*.

Our definition is a little less general than that in [Spi92] (p. 34) which has:

$$\left[\!\left[ S_0^{\mathbb{P}\, T_0} \upharpoonright S_1^{\mathbb{P}\, T_1} \right]\!\right] =_{df} [\![ (S_0 \wedge S_1) \upharpoonright T_1 ]\!]$$

Arguably our version is all that is required, since the standard definition permits $T_1$ to introduce new components and this is, perhaps, slightly odd for a *restriction* operation. In order to work with the standard definition we could use the general hiding operation over schema types in a manner similar to that employed below in Section 2.6, but we shall not give the details here.

The rules are then just a special case of those for conjunction. Using rules $(Z_\wedge)$ and $(Z_{\mathbb{P}\upharpoonright})$ we obtain the type rule:

$$\frac{\Gamma \rhd S_0 : \mathbb{P}\, T_0 \quad \Gamma \rhd S_1 : \mathbb{P}\, T_1 \quad T_1 \preceq T_0}{\Gamma \rhd S_0 \upharpoonright S_1 : \mathbb{P}\, T_1}$$

The introduction and elimination rules are then as follows:

$$\frac{\Gamma \vdash t \in S_0 \quad \Gamma \vdash t \upharpoonright T_1 \in S_1 \quad T_1 \preceq T_0}{\Gamma \vdash t \upharpoonright T_1 \in S_0 \upharpoonright S_1} \; (S^+_\upharpoonright)$$

This follows by rules $(S^+_\wedge)$ and $(\in_\upharpoonright)$, noting that $T_1 = T_1 \vee T_1$ and $T_0 = T_0 \vee T_1$.

$$\frac{\Gamma \vdash t \in S_0 \upharpoonright S_1 \quad \Gamma^-, y : T_0;\ \Gamma^+, y \in S_0, y \upharpoonright T_1 = t \vdash P}{\Gamma \vdash P} \; (S^-_{\upharpoonright_0}), \quad \text{for fresh } y$$

$$\frac{\Gamma \vdash t \in S_0 \upharpoonright S_1}{\Gamma \vdash t \in S_1} \; (S^-_{\upharpoonright_1})$$

These follow directly from the rules $(S^+_{\wedge_0})$, $(S_{\wedge_1})$ and $(\in^-_\upharpoonright)$ noting that $t^{\,T} = t \upharpoonright T$.

The substitution rule is:

$$\frac{\Gamma \vdash S_0^{\mathbb{P}\, T_0} = S_2 \quad \Gamma \vdash S_1^{\mathbb{P}\, T_1} = S_3 \quad T_1 \preceq T_0}{\Gamma \vdash S_0 \upharpoonright S_1 = S_2 \upharpoonright S_3}$$

## 2.6. Schema level hiding

Our basic hiding operation takes a single label (with its type) as an argument and, as we explained earlier, does duty for what, in other accounts, is a simple form of

schema existential quantification. In those accounts one also finds quantification over schemas in the category of schema expressions, for example [Spi92] (p. 76). We should provide, within our framework, a form of schema level hiding to correspond to this. This kind of operation has turned out to be of considerable value in the structuring of Z specifications. [WoD96] provides some excellent examples of operation *promotion* (*ibid.* chapter 13, see *e.g.* p. 187) which utilise this operation in order to promote an operation on a simple state to an operator on a global state of which it is a component. We shall return to this application in Section 4.

The definition is quite simple. In view of our earlier development we can define this easily using schema conjunction and restriction. Let $T_1 \preceq T_0$:

$$[\![S_0{}^{\mathbb{P}\,T_0} \setminus S_1{}^{\mathbb{P}\,T_1}]\!] =_{df} [\![(S_0 \wedge S_1) \upharpoonright (T_0 \setminus T_1)]\!]$$

Using rules $(Z_\wedge)$, $(Z_{\mathbb{P}\upharpoonright})$, together with the fact that $T_1 \preceq T_0$, we obtain the following type rule:

$$\frac{\Gamma \rhd S_0 : \mathbb{P}\,T_0 \quad \Gamma \rhd S_1 : \mathbb{P}\,T_1 \quad T_1 \preceq T_0}{\Gamma \rhd S_0 \setminus S_1 : \mathbb{P}(T_0 \setminus T_1)}$$

The introduction rule is calculated using rules $(S^+_\upharpoonright)$, $(S^+_\wedge)$ and the fact that $T_0 \setminus T_1 \preceq T_0$.

$$\frac{\Gamma \vdash t \in S_0 \quad \Gamma \vdash t \upharpoonright T_1 \in S_1 \quad T_1 \preceq T_0}{\Gamma \vdash t \upharpoonright (T_0 \setminus T_1) \in S_0 \setminus S_1}$$

The elimination rule is obtained using rule $(\in^-_\upharpoonright)$. Let $x$ be fresh and $\Gamma'$ be $\Gamma^-, x : T_0;\ \Gamma^+, x \in S_0, x \upharpoonright T_1 \in S_1, x \upharpoonright (T_0 \setminus T_1) = t$:

$$\frac{\Gamma \vdash t \in S_0{}^{\mathbb{P}\,T_0} \setminus S_1{}^{\mathbb{P}\,T_1} \quad \Gamma' \vdash P \quad T_1 \preceq T_0}{\Gamma \vdash P}$$

There is a useful equational rule for schema level hiding. This may be compared with the syntactic characterisation of (a simpler form of) schema existential quantification which is given in [WoD96] (p. 178). Let $\alpha D_1 = \{\cdots l_i \cdots\}$ and $\sigma = [\cdots l_i \cdots / \cdots z_i \cdots]$ where the $z_i$ are fresh variables.

$$\frac{\Gamma^- \rhd [D_0 \mid P_0] : \mathbb{P}\,T_0 \quad \Gamma^- \rhd [D_1 \mid P_1] : \mathbb{P}\,T_1 \quad [D_1] \preceq [D_0]}{\Gamma \vdash [D_0 \mid P_0] \setminus [D_1 \mid P_1] = [[D_0] \setminus [D_1] \mid \exists\, D_1\sigma \bullet (P_0 \wedge P_1)\sigma]}$$

The substitution rule is:

$$\frac{\Gamma \vdash S_0{}^{\mathbb{P}\,T_0} = S_2 \quad \Gamma \vdash S_1{}^{\mathbb{P}\,T_1} = S_3 \quad T_1 \preceq T_0}{\Gamma \vdash S_0 \setminus S_1 = S_2 \setminus S_3}$$

## 2.7. Definite description

Although definite descriptions nominally appear in Z as *terms* it is clear, from those sources which provide a logic for Z, that these terms must be understood to appear syncategorematically: the rules in [WoD96] and [Nic95] are expressed in terms of equality propositions, which we will write:

$$\mu x \in T \bullet P = t$$

Further evidence for this being the correct approach comes from [Spi88] in which the author remarks that (the meta-theory of) Z can be modelled within ZF set-theory *without* the axiom of choice. The salient point being that ZF with an *explicit* operator for definite descriptions (such as Hilbert's epsilon or Russell's iota) would *imply* the axiom of choice (see *e.g.* [Lei69]).

The characteristic formula for definite descriptions, $\mu x \in C \bullet P = t$, is translated into Z by means of:

$$\mu x \in C \bullet P = t =_{df} (\exists_1 x \in C \bullet P) \wedge P[x/t]$$

and then all references to such terms may be removed by the following contextual definition into Z. Let $z$ be a fresh variable:

$$P_0[z/\mu x \in C \bullet P_1] =_{df} \exists z \in C \bullet \mu x \in C \bullet P_1 = z \wedge P_0$$

Given the definition we easily obtain the following derived rules of typing and inference:

$$\frac{C : \mathbb{P} T \quad z : T \rhd P \ prop}{\mu z \in C \bullet P : T} \ (Z_\mu) \qquad \frac{\exists_1 z \in C \bullet P \quad t \in C \quad P[z/t]}{\mu z \in C \bullet P = t} \ (\mu^+)$$

$$\frac{\mu z \in C \bullet P = t}{t \in C} \ (\mu_0^-) \qquad \frac{\mu z \in C \bullet P = t}{P[z/t]} \ (\mu_1^-)$$

With definite description in place we have immediately a means for dealing with partial application. Given $f \in T_0 \nrightarrow T_1$ and $x \in T_0$ we can set:

$$f(x) =_{df} \mu y \in T_1 \bullet (x, y) \in f$$

and then rules for partial application follow from those above for definite description. This approach is unlikely to satisfy many, since the issue of reasoning with partial terms is a research topic which extends far beyond the study of Z (see *e.g.* [Jon95]) and within which controversy is the watchword. Indeed, the reliance upon the contextual (syntactically based) introduction of definite description does not satisfy us. Investigating this topic further is, however, not the thrust of our work here.

## 2.8. Conditional terms

With definite descriptions in hand we have a method for interpreting the *conditional terms* often employed in example Z specifications.

$$[\![ \mathit{if}\ P\ \mathit{then}\ t_0^T\ \mathit{else}\ t_1^T ]\!] =_{df} [\![ \mu x \in T \bullet (P \Rightarrow x = t_0) \wedge (\neg P \Rightarrow x = t_1) ]\!]$$

The following type assignment rule is then derivable using rules $(Z_\mu)$, $(Z_\wedge)$, $(Z_\Rightarrow)$, $(Z_=)$, $(Z_\neg)$ and lemmas 4.3($i$) and 4.6 from [HeR99]:

$$\frac{\Gamma \rhd P \ prop \quad \Gamma \rhd t_0 : T \quad \Gamma \rhd t_1 : T}{\Gamma \rhd \mathit{if}\ P\ \mathit{then}\ t_0\ \mathit{else}\ t_1 : T}$$

The following introduction rule can be derived, using the law of the excluded middle for the proposition $P$:

$$\frac{P \vdash t_0 = t_2 \quad \neg P \vdash t_1 = t_2}{(\mathit{if}\ P\ \mathit{then}\ t_0\ \mathit{else}\ t_1) = t_2}$$

## 2.9. Generic schemas

A generic schema is parameterised over one or more types. These are very easily accommodated within our regime. We will permit an extension of our language of types to include type variables:

$$T ::= \cdots \mid X$$

Then we may introduce a new category of generic schemas:

$$GS ::= S[X]$$

Finally we extend the language of schema expressions to include instantiated generic schemas:

$$S ::= \cdots \mid S[X := T]$$

Such instantiated schemas are interpreted into Z by means of:

$$[\![S[X := T]]\!] =_{df} [\![S]\!][X/T]$$

The following rules are then immediate:

$$\frac{P[\alpha[D]/t.\alpha[D]] \quad t \in [D][X/T]}{t \in [D \mid P][X := T]} \qquad \frac{t \in [D \mid P][X := T]}{P[\alpha[D]/t.\alpha[D]]} \qquad \frac{t \in [D \mid P][X := T]}{t \in [D][X/T]}$$

## 2.10. Alternative forms of quantification

There are several different forms of quantification which are adopted in the literature on Z. We shall, in this short section, only attempt to develop those alternatives presented in one of the normative sources: [Spi92]. The basic form of existential quantification ([Spi92] p. 70) is:

$$\exists S \bullet P$$

We shall interpret this by means of the following definitional extension, where $z$ is fresh:

$$[\![\exists S \bullet P]\!] =_{df} [\![\exists z \in S \bullet P[\alpha S/z.\alpha S]]\!]$$

As a consequence we would then induce the following rules:

$$\frac{S : \mathbb{P}T \quad z : T \rhd P[\alpha S/z.\alpha S]\ prop}{\exists S \bullet P\ prop} \qquad \frac{P[\alpha S/t.\alpha S] \quad t \in S}{\exists S \bullet P}$$

Let $y$ be a fresh variable in the following rule:

$$\frac{\Gamma \vdash \exists S \bullet P_0 \quad \Gamma^- \rhd S : \mathbb{P}T \quad \Gamma^-, y : T;\ \Gamma^+, y \in S, P_0[\alpha S/y.\alpha S] \vdash P_1}{P_1}$$

The basic forms for $\lambda$-expressions and definite descriptions in [Spi92] (p. 58) are:

$$\lambda S \bullet t$$

and:

$$\mu S \bullet P$$

We shall omit the translation of these (and their induced rules) since they follow the pattern we have just presented for the existential quantifier and are easily

calculated by analogy. The primitive form for set comprehension in [Spi92] (p. 57) is:

$$\{S \bullet t\}$$

We interpret this by means of the following definition. Let $z$ be a fresh variable:

$$[\![\{S \bullet t^T\}]\!] =_{df} [\![\{z \in T \mid \exists S \bullet z = t\}]\!]$$

We then obtain the following type assignment rule:

$$\frac{S : \mathbb{P}\, T_1 \quad z : T_1 \rhd t[\alpha S / z.\alpha S] : T_0}{\{S \bullet t\} : \mathbb{P}\, T_0}$$

The the introduction rule for this form of set comprehension is:

$$\frac{\Gamma^- \rhd S : \mathbb{P}\, T_1 \quad \Gamma \vdash t_2 \in S \quad \Gamma \vdash t_0 = t_1[\alpha S / t_2.\alpha S]}{\Gamma \vdash t_0 \in \{S \bullet t_1\}}$$

Let $y$ be a fresh variable in the following elimination rule:

$$\frac{\Gamma \vdash t \in \{S \bullet t_1\} \quad \Gamma^- \rhd S : \mathbb{P}\, T_1 \quad \Gamma^-, y : T_1;\ \Gamma^+, y \in S, t = t_1[\alpha S / y.\alpha S] \vdash P_1}{P_1}$$

## 2.11. The mathematical toolkit

We have said nothing about functions, sequences, bags *etc.* and the notation and operations which correspond to them. The reason for this is that these remaining features of Z are simple definitional extensions. As an example recall the standard definition of sequences. In our notation this would be:

$$\mathrm{seq}\, T =_{df} \{f \in \mathbb{N} \pfun T \mid \mathrm{dom} f = 1..\#f\}$$

which itself requires the development of finite partial functions; which in turn is defined (*e.g.* [Spi92] (p. 112)) in terms of partial functions *etc.* The corresponding display form $< \cdots >$ is of type $\mathbb{P}(\mathbb{N} \times T)$ and so is interpreted in terms of the display form for tuples. These details, and all the others, are completely covered in *e.g.* [Spi92].

## 2.12. Organisation of specifications

Z provides mechanisms for the overall organisation of specifications into paragraphs and sections. Where rules for these have been provided they have turned out to be among the most complex required, and are clearly the result of much ingenuity. In [Toy97] (p. 43) there are typechecking rules for sections and paragraphs which are enormously complex and which require extremely baroque side-conditions. At the very least these rules establish a useful basis for further work. The complications occurring in the rule for sections arise because each component induces a context which subsequent components inherit. It remains to be seen to what extent these complications are tamed by treating schema components as constants rather than variables. It is not even clear that these larger scale entities are best treated explicitly within the object logic itself. Their function is organisational rather than logical and the rules are more akin to side-condition

calculations (ensuring proper scoping and so on) than to *deduction*. This is clearly an important topic, especially in the context of the development of support tools, and perhaps best left to those engaged in that research area.

## 3. Priming and the $\theta$ operator

There are two operations, very commonly utilised in Z specifications, which we have, until now, avoided entirely. In this section we shall explain and demonstrate the mathematical problems which they, jointly, cause. Following this we will describe an alternative means by which the services they are meant to provide can be presented, with the added advantage that the formalisation is relatively simple, comprehensible and, consequently, usable.

There are two competing perspectives on schemas in Z, as they are currently understood, which are mutually incompatible. The older view is that a schema is a "piece of mathematical text" ([WoD96] p. 148) or the description of a state (*e.g.* [She95] p. 202). The more recent, dating roughly from the time when schemas became routinely used as sets and the $\theta$ operator was introduced, is that a schema is a "set of bindings" [WoD96] (p. 156) or "collection of possible values" [She95] (p. 199). The most striking example of this appears in [Dil94] (pp. 46–7), where, within two paragraphs, the author gives *both* accounts of schemas[2].

"$\cdots$ Schemas are used $\cdots$ to make precise what the *state space* of a given specification is. The state space is defined by means of a state schema." ([Dil94], Section 4.3.4, p. 46. Our emphasis.) "*PhoneDB* is the name of a schema which represents a before *state*. Decorating the name with a prime, for example *PhoneDB*′, represents the after *state*." ([Dil94], Section 4.3.4, p. 47. Our emphasis.)

The older perspective accounts for the use of schema priming: if $S$ is a schema representing the before state (singular), then $S'$ represents the after state. The notion of the $\Delta$-schema is paradigmatic of this view. It is somewhat surprising to discover that the $\Xi$-schema is paradigmatic of the alternative perspective. To see this we must first see what goes wrong when we attempt understand such schemas from the older perspective. Consider the schema $\Xi S =_{df} [\Delta S \mid \theta S = \theta S']$. It is very well-known that in the context of the definition $T =_{df} S'$ the schema $[S; T \mid \theta S = \theta T]$ is not even well-typed *a fortiori* not equal to $\Xi S$. But instead of tracing this unfortunate observation back to the root cause (the clash of perspectives we have introduced) a range of mathematically unpleasant manoeuvres have taken place in order to accommodate the situation. For example, in order to prevent Leibnitz's principle from failing one must ensure that the expression $\theta S'$ is not the application of $\theta$ to the schema $S'$ which ensures that the substitution is invalid. But this may not be enough: generally, $\theta$ may only be applied to schema *names*. This has the effect of making $\theta$ a non-extensional

[2] We realise, of course, that the range of text books need not necessarily reflect our best understanding of the subject, for which we might turn to [Spi88], [Spi92] and some sections of [Nic95]. The textbooks however fulfill a very different function here: they have been written over a substantial period of time, thus providing a historical perspective on Z and the evolution of its concepts. Additionally, they are an excellent repository of Z in practice. The practices and perspectives of the various textbooks constitutes an accurate reflection and record of the use of Z within the community of its practitioners over time. The more technical references, like our own, are to a greater or lesser degree attempts to normalise or to account for that practice. What we are highlighting here, in this section, is not an *inconsistency* which appears only within or between some informal accounts of Z, but a genuine *evolution* in the community's self-understanding of Z.

operation. These devices may prevent ambiguity and avoid the incoherence of a failure of Leibnitz's principle, but they do so by technical means which are complex and unwieldy, making formalisation extremely difficult and, even if achieved, of limited value. The problem is, rather, that the type of $S'$ is not the type required: we need the type of $S$ here. Indeed we do: the $\Xi$-schema is intended to link the initial state and the final state and these, under the *second* perspective, are both elements of $S$. It appears that the $\theta$ operation is inextricably linked with this second perspective. But from this viewpoint the $\Delta$-schema is incomprehensible, for it appears to suggest that operations change specifications of states (state spaces) rather than states. The solution to all this must begin by reconciling these pre-theoretic contradictions.

The older perspective, that schemas are states, is highly syntactic and it is linked with interpretations of the notation which are essentially based on macro-expansion. These have no, or very limited, mathematical properties. Moreover, this view is incompatible with almost all of the innovative work on Z which has taken place more recently, much of which has been introduced as a result of applying Z in practice. In particular, the greater role for schemas, as first-class entities, presupposes that they represent specifications of collections and not specifications of individuals. From this perspective it is easy to render the $\Xi$-schema by means of $[z, z' \in S \mid z = z']$ for some suitable choice of labels $z$ and $z'$. Note that it is now quite clear that $S$ describes the *set* of states over which the operation computes, and the before and after states both conform to that specification. As a result the type of the equality is preserved naturally, without resort to dubious technical tricks. The $\Delta$-schema is now best thought of as a declaration and not as a schema at all: $z, z' \in S$.

So far as the $\theta$ operation is concerned, we have not needed to employ it in the definition of the $\Xi$-schema because, instead of *including* a schema, we have introduced a declaration over the schema as a set. But this approach can be taken whenever the $\theta$ operation is normally required. It is a natural corollary of adding schemas as sets to Z in the systematic fashion we are advocating: the operation $\theta$ has no role to play.

## 3.1. Latent declarations

We have argued that we should remove the concepts of schema priming and the $\theta$ operator on both conceptual and mathematical grounds. We must then investigate whether or not the language remains expressive enough for its purposes. Certainly there is a change of style. Adapting existing Z specifications to our revised framework requires some care: when the $\theta$ operator is useful in standard Z we would introduce a declaration of schema type, where, most often, the standard Z would invoke a schema inclusion.

This approach, on its own, would require more explicit use of binding projection in specifications written in our system. Compare, for example, the following in standard Z ([WoD96], p. 175) and then our revised language.

$$BoxOffice =_{df} [seating : \mathbb{P}\, Seat, sold : Seat \pfun Customer \mid \mathrm{dom}\, sold \subseteq seating]$$

$$
\begin{array}{|l}
\hline
Return_0 \\
\Delta BoxOffice \\
s? : Seat \\
c? : Customer \\
\hline
s? \mapsto c? \in sold \\
sold' = sold \setminus \{s? \mapsto c?\} \\
seating' = seating \\
\hline
\end{array}
$$

$$BoxOffice =_{df} [seating \in \mathbb{P}\, Seat, sold \in Seat \pfun Customer \mid \mathrm{dom}\, sold \subseteq seating]$$

$$
\begin{array}{|l}
\hline
Return_0 \\
b, b' \in BoxOffice \\
s? \in Seat \\
c? \in Customer \\
\hline
s? \mapsto c? \in b.sold \\
b'.sold = b.sold \setminus \{s? \mapsto c?\} \\
b'.seating = b.seating \\
\hline
\end{array}
$$

The notational burden is rather similar to that one can encounter in programs which manipulate structured data. In Pascal, for example, one has the "with" idiom to aid presentation. A generalisation of this seems called for here.

We shall permit, as prime declarations, a new form which we will call *latent declarations*. These are written:

$$(l\xi \in)S$$

where $\xi$ is a, possibly absent, diacritical mark (prime, subscript *etc.*). Notice that we restrict the use of this idiom to *schemas* only: its purpose is to ameliorate the inexpressivity of our revision of Z which accrues because of the occasional replacement of schema inclusion by a declaration, and there is nothing to be gained by making it more general than absolutely necessary.

The idea is that one may, in the context of this declaration, refer to the components of $S$ directly. On the other hand, $l\xi$ is available, if necessary, when one would conventionally require the $\theta$-operator.

We can translate such a novelty into Z by means of:

$$[\cdots (l\xi \in) S^{\mathbb{P}\, T} \cdots \mid P] =_{df} [\cdots l\xi \in S \cdots \mid P[\alpha T\xi / l\xi.\alpha T]]$$

The diacritical mark $\xi$ plays a crucial role. It is perfectly possible (indeed highly likely in view of the inclusion of $\Delta$-schemas in operations) that a schema is effectively included twice in our version of Z. Consequently, these marks, which in standard Z refer to distinct components in distinct schemas, allow us to determine to which declaration the component belongs.

In the presence of this syntactic device we can write the schema above as:

*Return*$_0$

$(b, b' \in)BoxOffice$
$s? \in Seat$
$c? \in Customer$

---

$s? \mapsto c? \in sold$
$sold' = sold \setminus \{s? \mapsto c?\}$
$seating' = seating$

This is not significantly different from the standard presentation.

Additionally, we make use of the latently declared components at the same time as suppressing their appearance elsewhere. For example, in standard Z we might have ([WoD96] p. 193):

*Promote*

$\Delta Array$
$\Delta Data$
$index? : \mathbb{N}$

---

$index? \in \mathrm{dom}\, array$
$\{index?\} \lhd array = \{index?\} \lhd array'$
$array\ index? = \theta Data$
$array'\ index? = \theta Data'$

In our revised language this could now appear as:

*Promote*

$(a, a' \in)Array$
$(d, d' \in)Data$
$index? \in \mathbb{N}$

---

$index? \in \mathrm{dom}\, array$
$\{index?\} \lhd array = \{index?\} \lhd array'$
$array\ index? = d$
$array'\ index? = d'$

It is, perhaps, important to reinforce the point that our framework is likely to impose some differences in the *style* of specification. In particular, in evaluating our proposals with standard Z one must guard against assuming that simply transliterating existing specifications is the correct point of comparison. The following example demonstrates that one might approach a problem in quite a different way. The technique we shall illustrate is described in [Bow96] from which the example is adapted.

**Example 3.1.** The objective is to define a form of $\Xi$-schema which ensures that only some of the state components are invariant across a state change. Consider:

*S*

$a, b, c : \mathbb{N}$

Taking $\Delta S$ and $\Xi S$ as usual we define:

$$\Phi(z) =_{df} \Delta S \wedge (\Xi S \setminus [z : \mathbb{N}])$$

Calculation reveals that $\Phi(a) =$

$$\Delta S$$
$$b' = b$$
$$c' = c$$

In other words $\Phi(a)$ is the same as $\Xi S$ except that one component of $S$ (the component $a$) is *not* held invariant. Whereas we could represent this directly in our version of Z we might observe that the following is possible: $\Phi[X] =_{df}$

$$s, s' \in S$$
$$s \upharpoonright X = s' \upharpoonright X$$

Then we would represent the schema $\Phi(a)$ above as $\Phi[X := [b, c : \mathbb{N}]]$.

Although we might wish to argue that this is much clearer, this is not our purpose here. The point at issue is that it is a complex matter to determine the relative expressive merits of standard Z and our revision, because each language determines its own natural styles. This is well worth exploring in much more detail in the future. We shall make some further comments in Section 5.

## 4. Example

We shall not try to be over ambitious and will, by no means, attempt encyclopaedic coverage of Z specification techniques in this section. It will certainly remain to be seen whether or not what we have established as a revised Z meets the demands of practice. We would hope, at the very least, that the existence of a complete mathematical framework will encourage others to experiment.

Let us, at least, consider a reasonable example from the literature. This concerns the technique of *promotion* (see [WoD96] chapter 13). The example taken from this chapter (pp. 186-7) concerns the promotion of an operation over a local state to an operation over a global state. This is Z at its very best: providing a general organising strategy which structures a specification. First we present the example as it stands in the book.

*LocalScore*
$$s : \mathbb{P}\, Colour$$

*GlobalScore*
$$score : Players \pfun LocalScore$$

*AnswerLocal*
$$\Delta LocalScore$$
$$c? : Colour$$
$$s' = s \cup \{c?\}$$

*Promote*

$$\Delta GlobalScore$$
$$\Delta LocalScore$$
$$p? : Player$$

---

$$p? \in \operatorname{dom} score$$
$$\theta LocalScore = score\ p?$$
$$score' = score \oplus \{p? \mapsto \theta LocalScore'\}$$

Then the specification of *AnswerGlobal*, the operation over the global state, is given by promoting *AnswerLocal* with respect to *Promote*:

$$\exists \Delta LocalScore \bullet AnswerLocal \wedge Promote$$

In our presentation this would be rewritten as follows:

*LocalScore*

$$s \in \mathbb{P}\, Colour$$

*GlobalScore*

$$score \in Players \pfun LocalScore$$

*AnswerLocal*

$$(l, l' \in) LocalScore$$
$$c? \in Colour$$

---

$$s' = s \cup \{c?\}$$

*Promote*

$$(g, g' \in) GlobalScore$$
$$l, l' \in LocalScore$$
$$p? \in Player$$

---

$$p? \in \operatorname{dom} score$$
$$l = score\ p?$$
$$score' = score \oplus \{p? \mapsto l'\}$$

Then the specification of *AnswerGlobal* the operation over the global state is then given by:

$$(AnswerLocal \wedge Promote) \setminus [l, l' \in LocalScore]$$

What confidence can we have that the schemas we have defined are the intended interpretation? Since our operators are not defined by syntactic transformation we cannot undertake the simplification of [WoD96] p. 188 which demonstrates that $\exists \Delta LocalScore \bullet AnswerLocal \wedge Promote$ is equivalent to:

*AnswerGlobal*

$$\Delta GlobalScore$$
$$p? : Player$$
$$c? : colour$$

---

$$p? \in \operatorname{dom} score$$
$$\{p?\} \lhd score' = \{p?\} \lhd score$$
$$(score'\ p?).s = (score\ p?).s \cup \{c?\}$$

However, we have more or less the same apparatus in another guise: each of the syntactic transformations in the text-book have become instances of provable equalities in our Z logic. Putting together the various lemmas for the schema expressions from the technical development has established an equational logic for reasoning about schemas.

The first stage is to remove the latent declarations.

*AnswerLocal*

$$l, l' \in LocalScore$$
$$c? \in Colour$$

---

$$l'.s = l.s \cup \{c?\}$$

*Promote*

$$g, g' \in GlobalScore$$
$$l, l' \in LocalScore$$
$$p? \in Player$$

---

$$p? \in \operatorname{dom} score$$
$$l = g.score\ p?$$
$$g'.score = g.score \oplus \{p? \mapsto l'\}$$

Next, since our equations always require the $D^*$ form of declarations, we clearly have to use the rule $(\in^=)$ on *GlobalScore* since its declaration part is not of the right form. $GlobalScore =_{(\in^=)}$

*GlobalScore*$_0$

$$score \in \mathbb{P}(Players \times LocalScore)$$

---

$$score \in Players \pfun LocalScore$$

We can now substitute this for *GlobalScore* in *Promote*, and then, in turn, we can equate *Promote* with a schema whose declaration part is in the $D^*$ form.
$Promote =_{(sub,\in^=)}$

*Promote*$_0$

$$g, g' \in [score \in \mathbb{P}(Players \times [s \in \mathbb{P}\, Colour])]$$
$$l, l' \in LocalScore$$
$$p? \in Player$$

---

$$p? \in \operatorname{dom} score$$
$$l = g.score\ p?$$
$$g'.score = g.score \oplus \{p? \mapsto l'\}$$
$$g, g' \in GlobalScore_0$$

We now proceed to the conjunction:

$$AnswerLocal \land Promote =_{(sub)} AnswerLocal \land Promote_0 =_{(\land^=)}$$

$$\begin{array}{|l}
\textit{AG} \\
\hline
g, g' \in [score \in \mathbb{P}(Players \times [s \in \mathbb{P}\, Colour])] \\
l, l' \in LocalScore \\
c? \in Colour \\
p? \in Player \\
\hline
l'.s = l.s \cup \{c?\} \\
p? \in \operatorname{dom} g.score \\
l = g.score\ p? \\
g'.score = g.score \oplus \{p? \mapsto l'\} \\
g, g' \in GlobalScore_0 \\
\hline
\end{array}$$

Then, by substitution, we have

$$AnswerLocal \land Promote_0 \setminus [l, l' \in LocalScore] =_{(sub)} AG \setminus [l, l' \in LocalScore]$$

and then, by the equality rule for hiding, $AG \setminus [l, l' \in LocalScore] =_{(\setminus^=)}$

$$\begin{array}{|l}
AnswerGlobal_0 \\
\hline
g, g' \in [score \in \mathbb{P}(Players \times [s \in \mathbb{P}\, Colour])] \\
c? \in Colour \\
p? \in Player \\
\hline
\exists z, z' \in LocalScore \bullet \\
\quad z'.s = z.s \cup \{c?\} \land \\
\quad p? \in \operatorname{dom} g.score \land \\
\quad z = g.score\ p? \land \\
\quad g'.score = g.score \oplus \{p? \mapsto z'\} \land \\
\quad g, g' \in GlobalScore_0 \\
\hline
\end{array}$$

Note that $z'.s = z.s \cup \{c?\} \Leftrightarrow z' = \langle\!|\ s \Rrightarrow z.s \cup \{c?\}\ |\!\rangle$ is easily proved in the logic. So the predicate part of $AnswerGlobal_0$ is:

$$\begin{array}{l}
\exists z, z' \in LocalScore \bullet \\
\quad z' = \langle\!|\ s \Rrightarrow z.s \cup \{c?\}\ |\!\rangle \land \\
\quad p? \in \operatorname{dom} g.score \land \\
\quad z = g.score\ p? \land \\
\quad g'.score = g.score \oplus \{p? \mapsto z'\} \land \\
\quad g, g' \in GlobalScore_0
\end{array}$$

By the one-point rule, on the first equation, we have:

$$\begin{array}{l}
\exists z \in LocalScore \bullet \\
\quad p? \in \operatorname{dom} g.score \land \\
\quad z = g.score\ p? \land \\
\quad g'.score = g.score \oplus \{p? \mapsto \langle\!|\ s \Rrightarrow z.s \cup \{c?\}\ |\!\rangle\} \land \\
\quad g, g' \in GlobalScore_0
\end{array}$$

and again on the second equation gives:

$$p? \in \operatorname{dom} g.score \land$$
$$g'.score = g.score \oplus \{p? \mapsto \langlebar\, s \Rrightarrow (g.score\ p?).s \cup \{c?\} \,\rangle\!\rangle\} \land$$
$$g, g' \in GlobalScore_0$$

This then yields, by substitution:

*AnswerGlobal*$_2$

$$g, g' \in [score \in \mathbb{P}(Players \times [s \in \mathbb{P}\, Colour])]$$
$$c? \in Colour$$
$$p? \in Player$$

---

$$p? \in \operatorname{dom} g.score$$
$$g'.score = g.score \oplus \{p? \mapsto \langlebar\, s \Rrightarrow (g.score\ p?).s \cup \{c?\} \,\rangle\!\rangle\}$$
$$g, g' \in GlobalScore_0)$$

Now, using $(\in^{=})$ again (and this time from right to left) we can undo the manipulations on *GlobalScore* we began with:

*AnswerGlobal*$_3$

$$g, g' \in GlobalScore$$
$$c? \in Colour$$
$$p? \in Player$$

---

$$p? \in \operatorname{dom} g.score$$
$$g'.score = g.score \oplus \{p? \mapsto \langlebar\, s \Rrightarrow (g.score\ p?).s \cup \{c?\} \,\rangle\!\rangle\}$$

Rewriting the second equality using the same argument as [WoD96] we then have:

*AnswerGlobal*$_4$

$$g, g' \in GlobalScore$$
$$c? \in Colour$$
$$p? \in Player$$

---

$$p? \in \operatorname{dom} g.score$$
$$\{p?\} \lhd g'.score = \{p?\} \lhd g.score$$
$$(g'.score\ p?).s = (g.score\ p?).s \cup \{c?\}$$

Then re-introducing latent declarations, we finally obtain:

*AnswerGlobal*

$$(g, g' \in) GlobalScore$$
$$c? \in Colour$$
$$p? \in Player$$

---

$$p? \in \operatorname{dom} score$$
$$\{p?\} \lhd score' = \{p?\} \lhd score$$
$$(score'\ p?).s = (score\ p?).s \cup \{c?\}$$

This is precisely the natural transliteration of the *AnswerGlobal* schema which is given in [WoD96] into our version of Z. The equations we have developed and used are similar to the informal, purely linguistic, transformations adopted in the textbooks, and our logic explains why these work. In view of proposition 4.14 of [HeR99], deduction ensures type-correctness automatically. A logic, in fact,

permits more to be said about specifications, and then as a consequence, more can be checked, with a gain in confidence as a major benefit.

## 5. Conclusions and future work

The purpose of this and the companion paper [HeR99] was twofold. Most crucially, we wished to provide an analysis of the language Z within the context of a useful mathematical framework, thus establishing Z as a specification logic. A secondary aim has been a critique of the Z language which has become established in the literature. These two trajectories are linked. Whilst it would have been entirely possible to outline many of the conceptual conundrums which Z poses in a discursive style (and it must be said that almost everything we have said is known and shared by various workers in the Z research community), we have been determined to allow the mathematics to take the lead. As is very often the case, a mathematical approach does more than formalise; it additionally highlights areas of confusion and complexity. Consequently, we have used mathematical criteria to produce not only a formal account but a simple and (ultimately, we hope) usable account which retains the major benefits which Z offers: expressibility and scalability.

We have attempted to be reasonably comprehensive and have addressed, if in places only in outline, most of the major areas of the Z language. However, much remains to be done. We should like, in future publications, to develop and extend the work we have begun here on the schema calculus and, as we have mentioned, explore the organisation of specifications at the level of sections. In addition we wish to pursue program development in the context of the specification logic we have established. In particular, we are very interested in exploring other semantic foundations for Z based on a constructive, intensional set theory and to compare this with the traditional model, based as it is on classical, extensional set theory.

Finally, as we acknowledge in Section 3.1, our revised framework requires a significant change in *style* and a significant investigation in which existing strategies are re-expressed must be undertaken. The results of such an investigation must then be used to evaluate and modify our approach. Such an interplay between theory and practice is vital. It is also not clear how the revised language interacts with existing work on program development. From our point of view this is not a concern for, as we indicated in the previous paragraph, we aim to address this topic by replacing the standard classical, extensional model with an intensional and constructive model. However, there are clearly interesting avenues to explore which utilise more conventional mechanisms. In order to investigate any of the these topics deeply, it would be very useful to use the systems provided here as the basis for a proof development tool. Work on this has already begun [Völ98], though much remains to be achieved.

## Acknowledgements

We would like to thank the Department of Computer Science at the University of Waikato, New Zealand, the Centre for Discrete Mathematics and Theoretical Computer Science, New Zealand, the Royal Society of Great Britain, and the EPSRC (grant number GR/L57913) for financial assistance which has supported the development of this research. We are most grateful to Doug Goldson, Lindsay

Groves, Ian Toyn, Ray Turner and Mark Utting for many useful discussions. We are particularly grateful to the Editor-in-Chief, Cliff Jones, for his help and encouragement in the preparation of the final version of this paper. Finally, our thanks go to one of the referees, who performed the most thorough and fruitful task of refereeing that we have ever experienced.

## References

[BrM96] Brien, S. and Martin, A.: A tutorial of proof in standard Z. Technical report, Technical monograph PRG-120, University of Oxford, 1996.

[Bow96] Bowen, J.: *Formal specification and documentation using Z*. International Thompson Computer Press, 1996.

[Dil94] Diller, A.: *Z: An introduction to formal methods (2nd ed.)*. J. Wiley and Sons, 1994.

[HeR98] Henson, M. C. and Reeves, S.: A logic for the schema calculus. In J. P Bowen, Andreas Fett, and Michael G. Hinchey, editors, *11th Int. Conf. ZUM'98: the Z Formal Specification Notation*, LNCS 1493, pages 172–191. Springer-Verlag, 1998.

[HeR99] Henson, M. C. and Reeves, S.: Revising Z: Part I - logic and semantics. *Formal Aspects of Computing Journal*, 11(4): 359–380, 1999.

[Jon95] Jones, C. B.: Partial functions and logics: A warning. *Information Processing Letters*, 54(2):65–67, 1995.

[Lei69] Leisenring, A. C.: *Mathematical logic and Hilbert's ϵ-symbol*. McDonald Technical and Scientific, 1969.

[Mar98] Martin, A.: A revised deductive system for Z. Technical report, Technical Report TR98-21, SVRC, University of Queensland, 1998.

[Nic95] Nicholls, J. (ed.): *Z Notation: Version 1.2*. Z Standards Panel, 1995.

[She95] Sheppard, D.: *An introduction to formal specification with Z and VDM*. McGraw-Hill International, 1995.

[Spi88] Spivey, J. M.: *Understanding Z: A specification language and its formal semantics*. C.U.P., 1988.

[Spi92] Spivey, J. M.: *The Z notation: A reference manual*. Prentice Hall, 1992.

[Toy97] Toyn, I. (ed.): *Z Notation: Draft 0.8*. Unpublished draft, 1997.

[Völ98] Völker, N.: Private communication, 1998.

[WoD96] Woodcock, J. and Davies, J.: *Using Z: Specification, Refinement and Proof*. Prentice Hall, 1996.

*Received March 1998*
*Accepted in revised form April 1999 by C. B. Jones*

## Note added in proof

Since this paper was accepted for publication, the draft Z standard [Nic95] has been superseded by the Final Committee Draft Standard [Toy99]. See the note added in proof in [HeR99] for further details.

[Toy99] Toyn, I. (ed.): Z Notation: Final Committee Draft, CD 13568.2, ftp://ftp.york.ac.uk./hise_reports/cadiz/ZSTAN/fcd.ps.gz, 1999.