Investigating Z

MARTIN C. HENSON, *Department of Computer Science, University of Essex, Wivenhoe Park, Colchester, Essex CO4 3SQ, UK.*
E-mail: hensm@essex.ac.uk

STEVE REEVES, *Department of Computer Science, University of Waikato, Private Bag 3105, Hamilton, New Zealand.*
E-mail: stever@cs.waikato.ac.nz

Abstract

In this paper we introduce and investigate an improved kernel logic Z_C for the specification language Z. Unlike standard accounts, this logic is consistent and is easily shown to be sound. We show how a complete schema calculus can be derived within this logic and in doing so we reveal a high degree of logical organization within the language. Finally, our approach eschews all non-standard concepts introduced in the standard approach, notably object level notions of substitution and entities which share properties both of constants and variables. We show, in addition, that these unusual notions are derivable in Z_C and are, therefore, unnecessary innovations.

Keywords: Specification language Z, logic and semantics of specification languages.

1 Introduction

In this paper we introduce and investigate an improved kernel logic Z_C for the specification language Z, a logic in which, in particular, we can derive a schema calculus: a logic for the entire range of schema expressions permitted in Z.

The work presented here builds and improves upon our earlier work [7, 8] in several ways. Most importantly we have moved to a 'Church-style' from a 'Curry-style' presentation of both the language and the logic. This gives a simpler and more elegant account of the logic itself and of the technical development. The type coherence issues of [7], for example, are entirely absent from the system we give here. It would be reasonable to say that our previous approach made too many concessions to issues concerning the future implementation of the logic, something we feel we have rectified here with a gain in clarity. Secondly, [7, 8] mount a critique on Z, are openly revisionist and make several recommendations for *changing* certain key aspects of Z, using the logical analysis as a tool. In this paper, in contrast, we have kept close to Z as it is informally understood.[1] The reader will now be able to compare, in particular, our accounts of schema inclusion, θ-terms and Δ-schemas from two perspectives. Thirdly, some issues (such as schema composition and schema piping) are now treated in much fuller detail than in the past. Fourthly, the system Z_C given in this paper is much smaller than that given in [7]: several key notions which were basic for that system become derived in this paper (see especially Section 4). Finally, we spend some time showing how one of the innovations of the standard account, the so-called 'frogspawn' operators, are also analysable in Z_C and we are able to make a case that they are technically unnecessary.

Currently, the formalization of Z is given in [16] (version 1.4). Surprisingly, and unlike its

[1] Though with a few purely notational differences.

predecessors, this draft of the Z Standard does not contain a logic at all. Consequently we are unable to compare our work with it. The previous draft [12] (version 1.3) on the other hand does contain a logic, due essentially to Martin, which appears as Annex K. Unfortunately, and for reasons we do not know, this annex is not based on [9] (the most recent draft to appear before the publication of [12]) but on a much earlier and incomplete draft which contains many errors and notational inconsistencies. In view of this we take [9], rather than annex K of [12], to be the current definitive account of the standard logic.

The report [9] is an attempt to improve upon the earlier logic contained in the previous draft standard for Z [11] (version 1.2), [3] which suffered from a number of problems (inference did not preserve typing [7] and it was, in fact, inconsistent [5]). However, all versions of the standard logic, including the most recent, are very monolithic. [9] explicitly covers, or attempts to cover, *all* features of Z and does not reveal nor exploit any structure in the notions which make up the language. It makes no attempt at proof-theoretic economy: very many of the rules are redundant. Additionally, the system unexpectedly introduces *object*-level notions of substitution (supported by more than 70 equivalence axioms) and also unusual entities which resemble both variables and constants; this being technically supported by distinct notions of free variable and alphabet. This results in a large number of side-conditions on many of the rules. There is a semantics for Z given in [12] which, again, covers the entire language monolithically. There is no proof of soundness of the logic with respect to this interpretation in either [12] or [9].

Our approach in this paper is quite different. It reveals and exploits the logical structure of the language clearly. We first introduce a typed set-theory Z_C, which introduces *schema types*, and go on to show how the schema calculus of Z may be constructed as a series of conservative extensions of this basic system. The soundness of the entire logic for Z then reduces to the soundness of this simple, typed set-theory. This turns out to be trivial once we demonstrate how to model its schema types in ZF by means of a suitable dependent product space.

Concerning a fundamental conceptual matter, we feel that the account of the relationship between schema types and schema quantification we have given in this paper is a major improvement over the complications employed in earlier attempts to develop a satisfactory logic of Z. We believe that these complications have been largely responsible for problems with these earlier attempts. In addition, it must be better to proceed with conventional (and well understood) concepts of variable and constant than to introduce new entities with extraordinary properties.

Our touchstone in this paper is *logical compositionality*: as we introduce new linguistic constructs we ensure that they are defined in terms of the logical properties of their immediate constituents. Compositionality has always been a feature in Z semantics (see [12, 16]) and can be traced back to [13]. It has in our work, however, been raised to the logical level rather than left at the semantic level. Proofs of properties which characterize each new construct strictly adhere to the same principle: whenever a new construct is introduced, the proofs of its characteristic logical properties use those of the constructs directly used in its definition.

The main outcome of this methodology is a logic for the schema calculus, though we are also able to show that several non-standard ideas introduced in the standard approach are not needed.

2 The specification logic Z_C

In this section we shall describe a simple specification logic which we call Z_C. This is a typed set-theory based upon the notion of *schema type*. Our formulation in this paper will be 'Church-style', in which types are carried explicitly by the variables (and hereditarily by the terms). This contrasts with our earlier presentation [7] in which Z_C is presented 'Curry-style', with the types assigned by type assignment rules. There is, as a consequence of this style of presentation, a significant simplification in the presentation of the logic, and then consequent simplifications in the uses we make of it. One of the reasons why formalizations of Z have in the past been presented in Curry-style (our own included) is over-attention to mechanization: it is certainly true that an implementation of a logic for Z will require an explicit inference system, or algorithm, for type-checking. But this is essentially only an implementation and not a logical requirement. A logic must establish precisely its language, and in the case of Z this is most simply achieved in the manner here presented. Similarly, an implementation of a logic will have to determine syntactically well-formed expressions. In the case of Z this requires some form of type-checking in accordance with the logic. But how this is to be achieved is an entirely separate question.

2.1 The language of Z_C

Z_C is a *typed set theory*. We begin with the types:

$$T \ ::= \ \mathbb{U} \mid \mathbb{P}\, T \mid T \times T \mid [\cdots l : T \cdots].$$

Types of the form $\mathbb{U}$ are the names of *free types*. These may be given by equations of the form:[2]

$$\mathbb{U} \rightarrow \cdots \mid \ c_i \langle\!\langle \cdots \mathbb{U}_{ij} \cdots \rangle\!\rangle \mid \cdots$$

where any of the $\mathbb{U}_{ij}$ may be $\mathbb{U}$ (permitting recursion). In particular, j may be zero. An important example is:

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

This class of free types is quite simple, but has the virtues of covering many practical cases and of ensuring the existence of set theoretic models. In particular, we do not permit mutual recursion here, but the generalization is straightforward. More permissive notions of free type are possible and have been extensively analysed in [15], [1] and [2]. That work could be used to strengthen what we have given here if required, although we would insist on some proof-theoretically determined class of consistent types in order to maintain the overall integrity of the logic.

We will often permit the meta-variable D to range over sequences of type assignments such as $\cdots l_i : T_i \cdots$ (the order is not important), also writing $\alpha[D]$, when D is $\cdots l_i : T_i \cdots$, for the alphabet set (in the meta-language) of labels $\{\cdots l_i \cdots\}$. No label may occur more than once in such a type.

Types of the form $[D]$ are called *schema types*. Other operations on schema types that we will need: $[D_0] \sqsubseteq [D_1]$ holds when the set of type assignments in $[D_0]$ is a subset of those

[2] We use a rightfacing arrow rather than the standard '::=' for the introduction of free types in order to prevent a conflict with our (meta)-notation for the syntax of the system.

of $[D_1]$; $[D_0] \sqcup [D_1]$ is the schema type comprising all the type assignments appearing in the components. It is not defined, of course, if this union contains distinct type assignments for duplicated labels. $[D_0] \sqcap [D_1]$ is the schema type comprising all type assignments occurring in both $[D_0]$ and $[D_1]$. Finally, $[D_0] - [D_1]$ is the schema type comprising all type assignments in $[D_0]$ which are not in $[D_1]$.

All categories of the language of Z_C must be well-formed with respect to these types.

Next we have the category of *terms*: we assume the existence of a denumerable set of variables for each type T. We use t as a meta-variable over terms of arbitrary type. In addition, for notational clarity, we use the meta-variable C to range over sets (terms of type $\mathbb{P}\, T$) and S to range over *schemas* (those sets for which T is a schema type). The syntax of terms is then:

$$
\begin{array}{lcl}
t^{T} & ::= & x^{T} \mid \{z \in t^{T} \mid P\} \mid t^{[\cdots l: T \cdots]}.l \mid t^{T \times T_1}.1 \mid t^{T_0 \times T}.2 \\
t^{[D]} & ::= & t^{T_2} \upharpoonright [D] \\
t^{\mathbb{U}} & ::= & c_i \cdots t^{\mathbb{U}_{ij}} \cdots \\
t^{[\cdots l: T \cdots]} & ::= & \langle\!| \cdots l \Rrightarrow t^{T} \cdots |\!\rangle \\
t^{T_0 \times T_1} & ::= & (t^{T_0}, t^{T_1}) \\
C^{\mathbb{P}\,\mathbb{U}} & ::= & \mathbb{U}^* \\
C^{\mathbb{P}\,T} & ::= & \mathbb{P}\, t^{T} \\
C^{\mathbb{P}(T_0 \times T_1)} & ::= & C^{\mathbb{P}\,T_0} \times C^{\mathbb{P}\,T_1} \\
C^{\mathbb{P}[\cdots l: T \cdots]} & ::= & [\cdots l \in C^{\mathbb{P}\,T} \cdots]
\end{array}
$$

where $[D] \sqsubseteq T_2$.

We pronounce the symbol $\upharpoonright$ 'filter' and the purpose of filtered terms is to permit the restriction of bindings to a given schema type. These are crucial for establishing the logic for the schema calculus. We immediately introduce two notational conventions in order to avoid the repeated use of filtering in the context of membership and equality propositions. Note that the T in $t \upharpoonright T \in C$, for example, is unnecessary since it can always be recovered from the type of C (there is only one T for which the proposition is type correct). So, we can abbreviate the above restricted membership statement by leaving out the type, although we must indicate the fact that a restriction takes place, if only in order to differentiate from a non-well-typed membership statement. We, therefore, introduce the following definition.

DEFINITION 2.1

$$t^{T_0} \dot{\in} C^{\mathbb{P}\,T_1} =_{df} t \upharpoonright T_1 \in C \qquad (T_1 \sqsubseteq T_0).$$

A similar tactic can be employed with equality.

DEFINITION 2.2

$$t_0^{T_0} \doteq t_1^{T_1} =_{df} t_0 \upharpoonright (T_0 \sqcap T_1) = t_1 \upharpoonright (T_0 \sqcap T_1) \qquad (T_1 \sqsubseteq T_0 \ or \ T_0 \sqsubseteq T_1).$$

As usual we can write $t(x)$ to indicate a free variable of the term t. For terms of schema type we shall need a more complex notational convention to indicate situations in which a binding denotes a term at a particular label. Specifically, suppose that t has the schema type $[\cdots l_i : T_i \cdots]$. We shall write $t(\cdots l_i \Rrightarrow t_i \cdots)$ to indicate that, in particular, $\cdots t.l_i = t_i \cdots$.

Sets, then, are formed from the free types by powerset, Cartesian product, schema sets and separation (bounded comprehension). We will often write schema sets as $[E]$ (where

E ranges over sequences of the form $\cdots l_i \in C_i \cdots$) by analogy with schema types and generalize the alphabet operator in the obvious manner. Among the sets are the *carriers* of the types. These are formed by closing the carriers for the basic types $\mathbb{U}^*$ under the three set-forming operations with corresponding operations in the type language. In the sequel we will often write T as a set (the carrier of the type T). In this regard we are following the notational abuse described in [14] (p. 24); this is entirely harmless since only a type can appear as a superscript, and only a set can follow the membership relation.

The formulæ of Z_C delineate a typed bounded predicate logic.

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

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

A crucial observation is *unicity of types*: every term and set of Z_C has a unique type. We can make great use of this observation. It enables us to remove type decoration in most circumstances and leads to a very elegant and simple presentation of the system and its extensions, much more so than in other approaches [7].

2.2 The logic of Z_C

The judgements of the logic have the form $\Gamma \vdash_C P$ where Γ is a set of formulæ.

The logic is presented as a natural deduction system *in sequent form*. When, later, we need to undertake derivations in the logic, we shall present them in *pure* natural deduction form, in which a sequent of the form $\cdots P_i \cdots \vdash P$ will be represented by:

$$\cdots \quad \begin{array}{c} P_i \\ \hline \vdots \\ P \end{array} \quad \cdots$$

We shall omit all data (entailment symbol, contexts, type etc.) which remains unchanged by a rule. In the rule ($\exists^-$), the variable y may not occur in C, P_0, P_1 nor any other assumption.

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

$$\frac{P \vdash \bot}{\neg P}\ (\neg^+) \qquad \frac{P \quad \neg P}{\bot}\ (\bot^+) \qquad \frac{\neg\neg P}{P}\ (\neg^-) \qquad \frac{\bot}{P}\ (\bot^-)$$

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

$$\frac{}{\Gamma, P \vdash P}\ (ass) \qquad \frac{}{t = t}\ (ref) \qquad \frac{t = t' \quad P[z/t]}{P[z/t']}\ (sub)$$

$$\frac{}{\langle\!|\cdots l_i \Rrightarrow t_i \cdots |\!\rangle.l_i = t_i}\ (\Rrightarrow^{=}_{\mathrm{o}}) \qquad \frac{}{\langle\!|\cdots l_i \Rrightarrow t.l_i \cdots |\!\rangle = t^{[\cdots l_i \in T_i \cdots]}}\ (\Rrightarrow^{=}_{\mathrm{1}})$$

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

$$\frac{P[z/t] \quad t \in C}{t \in \{z \in C \mid P\}}\ (\{\}^{+}) \qquad \frac{t \in \{z \in C \mid P\}}{t \in C}\ (\{\}^{-}_{\mathrm{o}}) \qquad \frac{t \in \{z \in C \mid P\}}{P[z/t]}\ (\{\}^{-}_{\mathrm{1}})$$

$$\frac{z \in C_0 \vdash z \in C_1}{C_0 \in \mathbb{P}\, C_1}\ (\mathbb{P}^{+}) \qquad \frac{C_0 \in \mathbb{P}\, C_1 \quad t \in C_0}{t \in C_1}\ (\mathbb{P}^{-})$$

$$\frac{t_0 \in C_0 \quad t_1 \in C_1}{(t_0, t_1) \in C_0 \times C_1}\ (\times^{+}) \qquad \frac{t \in C_0 \times C_1}{t.1 \in C_0}\ (\times^{-}_{\mathrm{o}}) \qquad \frac{t \in C_0 \times C_1}{t.2 \in C_1}\ (\times^{-}_{\mathrm{1}})$$

$$\frac{\cdots z_{ij} \in \mathbb{U}_{ij} \cdots}{c_i \cdots z_{ij} \cdots \in \mathbb{U}}\ (\mathbb{U}^{+}) \qquad \frac{\cdots z_{ij} \in \mathbb{U}_{ij} \cdots \quad \cdots z_{kl} \in \mathbb{U}_{kl} \cdots}{c_i \cdots z_{ij} \cdots \neq c_k \cdots z_{kl} \cdots}\ (\mathbb{U}_{\neq})$$

$$\frac{c_i \cdots z_{ij} \cdots = c_i \cdots y_{ij} \cdots}{z_{ij} = y_{ij}}\ (\mathbb{U}_{=})$$

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

where the y_k are all those variables occurring in the z_{ij} with type $\mathbb{U}$.

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

$$\frac{t_0 \equiv t_1}{t_0 = t_1}\ (ext) \qquad \frac{t^{T}.l_i = t_i \quad [\cdots l_i : T_i \cdots] \sqsubseteq T}{(t \restriction [\cdots l_i \in T_i \cdots]).l_i = t_i}\ (\restriction^{=})$$

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 Z_C is extensional.

As an example of the rules for free types we can give the following specializations for $\mathbb{N}$, as defined in Section 2.1 above:

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

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

PROPOSITION 2.3
The following axiom is admissible for Z_C:

$$\frac{}{t^T \in T} \quad (T)$$

PROOF. By induction on the structure of the term t. For example:
Case: $t = (t_0, t_1)^{T_0 \times T_1}$:
We have to show that $(t_0, t_1) \in T_0 \times T_1$ and we may assume, *ex hypothesi*, that $t_i \in T_i$ $(i \in 2)$. This follows immediately by rule $(\times^+)$. ∎

This is such a fundamental relationship between the type theory and set theory of Z_C that we shall, from now on, take Z_C to incorporate this axiom. Similarly, the following weakening rule is admissible and is incorporated within the system.

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

2.3 Partial terms

The language of Z permits terms which do not denote, in particular definite descriptions and partial applications. The latter are subsumed by the former, so we will restrict our attention to definite descriptions. Informal use of Z makes it clear that membership and equality are weak: $t \in C$ does not imply that t is defined. However, it remains an open question as to whether a logic should be silent regarding definedness or incorporate a predicate for definedness of terms.

For example the former position is taken in [18] where rules for definite descriptions introduce a deliberate incompleteness into the logic for those terms which do not uniquely characterize a value. Their rules, and as a consequence their treatment of partial application, could be interpreted within Z_C directly. We adopted a related approach to definite description in [7].

The alternative is to introduce an atomic predicate of the form $t{\downarrow}$, asserting that t is defined, and a corresponding logic of weak membership and equality. This may, or may not, be a fruitful approach to practical reasoning and, because it needs careful and extensive investigation, we leave its exploration for the future.

3 A model of Z_C in ZF

In this section we provide an interpretation from the language of Z_C into ZF and prove the soundness of the Z_C logic. The central idea is our interpretation of schema types as dependent

products over a family of sets from a small (in ordinal terms) cumulative universe. In what follows we suppose ourselves to working within any model $\langle M, \in \rangle$ of ZF. We begin with free types. Recall the general scheme for a free type $\mathbb{U}$:

$$\mathbb{U} \to \cdots \mid \; c_i \langle\!\langle \cdots \mathbb{U}_{ij} \cdots \rangle\!\rangle \mid \cdots$$

First we define for each summand of $\mathbb{U}$ a function $\mathcal{U}_i$:

$$\mathcal{U}_i(Y) =_{df} \{c_i\} \times \cdots \times A_{ij} \times \cdots$$

where $A_{ij} = Y$ when $\mathbb{U}_{ij} = \mathbb{U}$ and $[\![\mathbb{U}_{ij}]\!]$ otherwise. Then we can associate with $\mathbb{U}$ a function $\mathcal{U}$ defined so that

$$\mathcal{U}(Y) =_{df} \bigcup_i \mathcal{U}_i(Y).$$

Then the interpretation of $\mathbb{U}$ in ZF is given by:

$$[\![\mathbb{U}]\!] = \bigcup_{n \geq 0} \mathcal{U}^n(\{\}).$$

The following definition in ZF constructs a tiny (in ordinal terms) cumulative hierarchy.

DEFINITION 3.1

$$\begin{array}{llcl} (i) & F(0) & = & \bigcup \mathbb{U} \\ (ii) & F(\alpha + 1) & = & F(\alpha) \cup \mathbb{P}\, F(\alpha) \\ (iii) & F(\omega) & = & \bigcup_{\alpha < \omega} F(\alpha). \end{array}$$

This function is guaranteed to exist by transfinite induction (in fact only transfinite induction below $\omega.2$ is required) and we then take $F(\omega + 1)$ to be the universe within which the type system of Z may be interpreted. This universe is a *set*.

Let $B(X)$ be an I-indexed family of sets over $F(\omega)$ (that is, $B(X) \in I \to F(\omega + 1)$). Then we can define a *dependent function space* which is suitable for our purposes as follows:

$$\Pi_{(X \in I)}.B(X) = \{f \in I \to F(\omega) \mid (\forall\, i \in I)(f(i) \in B(i))\}.$$

This we can harness to interpret the schema sets (consequently also the schema types) of Z_C. We will write $E^\star$ for the interpretation in the model of a Z_C entity E.

$$[\cdots l_i : C_i \cdots]^\star = \Pi_{(X \in I)}.B(X)$$

where $I = \{\cdots l_i \cdots\}$ and $B(l_i) = C_i^\star$. Each free type $\mathbb{U}$ is interpreted as follows:

$$\mathbb{U}^\star = \{x \in F(0) \mid x \in [\![\mathbb{U}]\!]\},$$

which is a set in $F(1)$. The remaining types of Z are mapped by $[\![\ \_\ ]\!]$ to ZF in the obvious way.

The labels l_i can be modelled in ZF in any number of ways, for example as finite ordinals. The only important point is that they be formally distinguishable from one another. We shall write them in ZF as we do in Z_C for simplicity.

With this in place we can easily interpret the binding projection terms of the form $t.l$ by application in ZF, i.e. as: $t^\star\ l$. Finally, the bounded quantification in Z_C can be unpacked in ZF in the standard manner.

PROPOSITION 3.2 (Soundness of Z_C types)
For every Z_C term $t^{\,T}$ we have $\vdash_{ZF} t^\star \in T^\star$.

PROPOSITION 3.3 (Soundness of Z_C logic)
If $\Gamma \vdash_C P$ then $\Gamma^\star \vdash_{ZF} P^\star$.

4 Schema calculus fundamentals

In this section, and the sections which follow, we provide interpretations for a sequence of extended languages. To simplify the presentation we shall adopt the convention of not stating any clauses in such an interpretation which are merely homomorphic. All interpretations are indicated using heavy brackets (i.e. $[\![\ \_\ ]\!]$); each such interpretation maps an extended language into the language of the previous extension of Z_C (for example the interpretation of Section 4.1 maps Z_C + *Schema* into Z_C and that of Section 4.3 maps Z_C + *SchemaExpressions* into Z_C + *Schema* + *Theta*). Indeed, in the previous section we used the heavy brackets to map Z_C to ZF.

To begin this we first demonstrate how a core schema calculus can be derived from the kernel system Z_C. Once this is in place we will be able to extend the calculus still further in order to provide a logic for the entire range of schema expressions allowed in Z.

4.1 Introducing schemas

As we have already indicated, schemas are simply sets of type $\mathbb{P}\,T$ when T is a schema type; in other words, they are *sets of bindings*. As it stands, our only means for introducing schemas is by means of schema sets of the form $[E]$. We now give an extended language of terms to permit the conventional notation for schemas:

$$S^{\mathbb{P}\,T} \ ::= \ \cdots \mid [S^{\mathbb{P}\,T} \mid P].$$

As is usual, we will write schemas of the form: $[[E] \mid P]$ as $[E \mid P]$ and we allow the obvious generalization of our alphabet operator to schemas: $\alpha[S \mid P] = \alpha S$. Note that the meta-variable P, which occurs in the schema, can range over the propositions *extended with labels as terms*. We must explain precisely how ordinary substitution is to be extended here. The only clauses of significance are

$$\begin{array}{lll} (i) & \langle\!| \cdots l_i \Rrightarrow t_i \cdots |\!\rangle[l/t] & = \ \langle\!| \cdots l_i \Rrightarrow t_i[l/t] \cdots |\!\rangle, \\ (ii) & t.l_1[l_0/t_1] & = \ t[l_0/t_1].l_1, \end{array}$$

for any label l_0 (including l_i and l_1). We shall need to indicate multiple substitutions for all labels in a given alphabet, writing $[\alpha[D]/t.\alpha[D]]$ to indicate the family of substitutions $\cdots[l_i/t.l_i]\cdots$ when t is some term and $\alpha[D] = \{\cdots l_i \cdots\}$. The right-hand-side is, of course, syntactically valid in Z_C. The interpretation of schemas in Z_C is then given by

DEFINITION 4.1
Let Z be a fresh varible.

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

Given this definition we may provide the following rules using the schema notation.

PROPOSITION 4.2
The following rules are sound for the interpretation of schemas in Z_C:

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

PROOF. Trivial: these are all special cases of the three rules for bounded comprehension in Z_C. ∎

4.2 θ-expressions

In view of the extended language of terms we permit in schemas, we may make the usual identification of θ-expressions with *characteristic bindings*:

$$\theta S^{\mathbb{P}[\cdots l_i : T_i \cdots]} =_{df} \langle\!| \cdots l_i \Rrightarrow l_i \cdots |\!\rangle.$$

It is, perhaps, worth explaining in detail how this definition interacts with the definition of schemas we have given. Consider a schema $[S \mid P]$ in which θS occurs in P. This schema is interpreted in Z_C by the comprehension $\{z \in [\![S]\!] \mid [\![P[\alpha S / z.\alpha S]]\!]\}$. An occurrence of θS in P has, consequently, the Z_C interpretation $\langle\!| \cdots l_i \Rrightarrow z.l_i \cdots |\!\rangle$ in the Z_C proposition $[\![P[\alpha S / z.\alpha S]]\!]$. In view of axiom $(\Rrightarrow^{=}_{1})$, this means that, in this context, θS has the Z_C interpretation z.

There are well-known complications in Z concerning the use of *primed* schemas and the θ operator. In other publications we have tried to explain that these result from a deep rooted ambiguity regarding the nature of schemas [8]. Here, however, we adhere to the standard informal understanding of Z to provide a point of comparison with that earlier work. Essentially, the key point is that terms of the form $\theta S'$ should be regarded as operations on S and not on S'. It might then be wiser to accept this explicitly as indicating a distinct operation by writing such expressions as $\theta' S$. In any event, notational niceties notwithstanding, such expressions are defined by:

$$\theta' S^{\mathbb{P}[\cdots l_i : T_i \cdots]} =_{df} \langle\!| \cdots l_i \Rrightarrow l'_i \cdots |\!\rangle.$$

Consequently, occurrences of $\theta' S$ (or, equivalently: $\theta S'$) in the context of a schema $[S' \mid P]$, itself interpreted as $\{z \in [\![S']\!] \mid [\![P[\alpha S' / z.\alpha S']]\!]\}$, will be interpreted in Z_C as $\langle\!| \cdots l_i \Rrightarrow z.l'_i \cdots |\!\rangle$ (which, *nota bene*, is *not* equivalent to z).

4.3 Schema expressions

We may now introduce the schema expressions upon which the schema calculus is based. First of all we extend our syntax of schemas with the basic operations of disjunction, negation, and existential hiding.[3]

$$S \ ::= \ \cdots \mid S \vee S \mid \neg S \mid \exists\, l \in T \bullet S.$$

[3]We are grateful to one of our referees for pointing out that, historically, the first schema calculus operation to be introduced was conjunction, which corresponds to the product operator in relational databases. Further, the name 'schema' was taken from precisely this field. However, we have followed a treatment which is not uncommon in logical presentations, which is to introduce $\neg$, $\vee$ and $\exists$ as fundamental and then define the other logical operators in terms of them.

We first define three algebraic operations in Z_C which correspond to this extended language. These are naturally polymorphic: the type of their instances being determined by the particular sets to which they are applied.

DEFINITION 4.3
Let Z be a fresh variable.

$$\begin{array}{lll}
(i) & \neg C^{\mathbb{P}\, T} & =_{df} \{z \in T \mid z \notin C\} \\
(ii) & C_0^{\mathbb{P}\, T_0} \vee C_1^{\mathbb{P}\, T_1} & =_{df} \{z \in T_0 \sqcup T_1 \mid z \upharpoonright T_0 \in C_0 \vee z \upharpoonright T_1 \in C_1\} \\
(iii) & \exists\, l \in T_0 \bullet C^{\mathbb{P}\, T_1} & =_{df} \{z \in T_1 - [l : T_0] \mid \exists\, x \in T_1 \bullet \\
 & & \quad x \in C \wedge z = x \upharpoonright (T_1 - [l : T_0])\}.
\end{array}$$

With these in place it is possible to interpret the new language of schema expressions in Z_C.

DEFINITION 4.4

$$\begin{array}{lll}
(i) & [\![\neg S]\!] & =_{df} \neg\, [\![S]\!] \\
(ii) & [\![S_0 \vee S_1]\!] & =_{df} [\![S_0]\!] \vee [\![S_1]\!] \\
(iii) & [\![\exists\, l \in T \bullet S]\!] & =_{df} \exists\, l \in T \bullet [\![S]\!]\,.
\end{array}$$

The most important observation is the *compositional/algebraic* nature of these definitions. This is possible because the language is *typed* and the concrete structure which is necessary to effect the interpretation is carried explicitly by these types.

What we call *existential hiding* is usually referred to either as schema existential quantification or simply as hiding (with an alternative notation). The use of schema components as variables in the context of quantifiers (such as the existential here) but as constants in, for example, bindings has led to unusual complications: for example, in the sketch of a logic in [11]. This leads to the suggestion that Z declarations introduce entities which are *variables of some new species*. This is a trap that has dogged efforts to develop satisfactory formal models of Z in the past. Our approach here, and elsewhere, introduces no such unwelcome entities: declarations introduce *constants* and schemas are a special notation for sets of bindings, this latter desideratum being entirely uncontested in the literature. We shall make further comments about the use of quantification in the schema calculus in Section 4.6 below.

4.4 Renaming and priming

In addition to the three basic schema expressions, we have what is known in the literature as *renaming*. This permits a systematic substitution for labels. Let l_1 be a fresh label.

$$\frac{t \in S}{t[l_0 \leftarrow l_1] \in S[l_0 \leftarrow l_1]}\ (S^{+}_{\leftarrow}) \qquad \frac{t \in S[l_0 \leftarrow l_1]}{t[l_1 \leftarrow l_0] \in S}\ (S^{-}_{\leftarrow})$$

All occurrences of labels are systematically replaced, for example:

$$\begin{array}{lll}
(i) & \langle\!| \cdots l_i \Rrightarrow t_i \cdots |\!\rangle[l \leftarrow m] & = \langle\!| \cdots l_i[l \leftarrow m] \Rrightarrow t_i[l \leftarrow m] \cdots |\!\rangle \\
(ii) & t.l_0[l_1 \leftarrow m] & = t[l_1 \leftarrow m].l_0[l_1 \leftarrow m] \\
(iii) & l[l \leftarrow m] & = m \\
(iv) & l_0[l_1 \leftarrow m] & = l_0 \qquad \text{when } l_0 \neq l_1\,.
\end{array}$$

Priming can then, as usual, be understood as an instance of renaming. When S has the type $\mathbb{P}[\cdots l_i : T_i \cdots]$ we have:

$$S' =_{df} S[\cdots l_i \leftarrow l'_i \cdots].$$

It is worth highlighting that priming, in particular, sets up a *logical relationship* between certain labels. This is not difficult to achieve in our approach because the labels are *constants*. In this regard consider the logically similar relationship between names and co-names in CCS [10]: again, these names are constants. This contrasts with the standard approaches in which these entities are *variables*, a technical decision which we discussed in the previous section.

4.5 The logic of the schema calculus

We now provide a sound logic for schema expressions. In most instances we will not give the proofs in full, relying on an indication of the major rules involved for guidance. A few proofs are given in full for illustration. Beginning, then, with negation:

PROPOSITION 4.5
The following rules for schema negation are sound for the interpretation in Z_C:

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

These rules are unexpectedly efficient: the fact that the language is typed ensures that a judgement of the form $t \notin S$ (for example) requires t at the very least to be *potentially* a member of S. As a consequence we may conclude immediately that $t \in \neg S$.

PROOF. Using $(\{\}^+)$ and $(\{\}^-_1)$. ∎

Moving on to disjunction.

PROPOSITION 4.6
The following three rules are sound for the interpretation of schema disjunction in Z_C:[4]

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

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

PROOF. These are justified by means of the following derivations in Z_C:

$$\dfrac{\dfrac{t \upharpoonright T_0 \in S_0}{t \upharpoonright T_0 \in S_0 \vee t \upharpoonright T_1 \in S_1}\ \vee^+_0 \quad \dfrac{}{t \in T_0 \sqcup T_1}\ (T)}{t \in \{z \in T_0 \sqcup T_1 \mid z \upharpoonright T_0 \in S_0 \vee z \upharpoonright T_1 \in S_1\}}\ (\{\}^+)$$

[4]Here, and in many other places we have omitted type information for simplicity of presentation. The (meta-) types of the (meta-) variables occurring in the rules are uniquely determined by considering (all) of its constituent sequents.

A similar derivation justifies the remaining introduction rule, and for the elimination rule we have:

$$\dfrac{\dfrac{t \in \{z \in T_0 \sqcup T_1 \mid z \upharpoonright T_0 \in S_0 \vee z \upharpoonright T_1 \in S_1\}}{t \upharpoonright T_0 \in S_0 \vee t \upharpoonright T_1 \in S_1}\ (\{\}_1^-) \qquad \begin{array}{c}\overline{t \upharpoonright T_0 \in S_0} \\ \vdots \\ P\end{array} \qquad \begin{array}{c}\overline{t \upharpoonright T_1 \in S_1} \\ \vdots \\ P\end{array}}{P}\ (\vee^-)$$

The following two rules are direct consequences of these schema operations.

COROLLARY 4.7

$$\dfrac{}{t \in S \vee \neg S}\ (S_{LEM}) \qquad \dfrac{t \in S \quad t \in \neg S}{\bot}\ (S_{CON})$$

Finally we have existential hiding.

PROPOSITION 4.8
The following two rules are sound for the interpretation of schema existential hiding in Z_C:

$$\dfrac{t \in S}{t \mathrel{\dot\in} \exists\, l \in T \bullet S}\ (S_\exists^+)$$

$$\dfrac{t \in \exists\, l \in T \bullet S \quad y \in S, y \doteq t \vdash P}{P}\ (S_\exists^-)$$

Where y does not occur in S, P, t nor any other assumptions; in other words the usual side-conditions for eigenvariables apply.

PROOF. Using $(\{\}^+)$, $(\exists^+)$ and $(\{\}^-)$, $(\exists^-)$ respectively. ∎

4.6 Equational logic for the schema calculus

It is now possible to show that the expected equational relationships to be found in the literature [18], and which are commonly used to *define* the schema expressions, are provable *in the schema logic*. It is important to note that the proofs which follow are all undertaken strictly in the logic for schemas which we have just introduced. This ensures that the logic is *adequate* and demonstrates that *the proofs are compositional*: membership in a compound expression is determined solely in terms of the membership conditions on the proper components. This definitional and logical compositionality inherent in our approach should be contrasted with the highly non-compositional approach of *defining* schema operations by means of these equations [18].

In the proofs which follow we will make use of simple consequences of pure predicate logic (such as De Morgan equivalences) without further comment.

PROPOSITION 4.9

$$\neg[D \mid P] = [D \mid \neg P].$$

PROOF. ($\subseteq$): ($S^-_\neg$) and ($\vee^-$). ($\supseteq$): ($S^+_\neg$). ∎

PROPOSITION 4.10

$$[D_0 \mid P_0] \vee [D_1 \mid P_1] = [D_0] \sqcup [D_1] \mid P_0 \vee P_1].$$

PROOF. ($\subseteq$): ($S^-_\vee$) and ($\vee^+_{\text{o}}$) and ($\vee^+_{\text{1}}$). ($\supseteq$): ($S^+_{\vee_{\text{o}}}$), ($S^+_{\vee_{\text{1}}}$) and ($\vee^-_{\text{o}}$). ∎

PROPOSITION 4.11

$$\exists\, l \in T \bullet [D \mid P] = [D] - [l \in T] \mid \exists\, z \in T \bullet P[l/z]]$$

where z is fresh.

PROOF. We will write $[D^{-l}]$ for $[D] - [l \in T]$ and assume that its form is $[\cdots l_i \in T_i \cdots]$. We further write t' for the binding $\langle\!|\ \cdots l_i \Rrightarrow t.l_i \cdots, l \Rrightarrow y\ |\!\rangle$ and the three substitutions $[\alpha[D^{-l}]/t.\alpha[D^{-l}]]$, $[\alpha[D]/t'.\alpha[D]]$ and $[\alpha[D^{-l}]/y \upharpoonright [D^{-l}].\alpha[D^{-l}]]$ as σ_0, σ_1 and σ_2. Note, in particular, that $P\sigma_1 = P\sigma_0[l/y]$. The schema $[D \mid P]$ we will write as S and, finally, the equation in question will be written $S_l = S_r$. The result follows, by rule (ext), from these two derivations.

$$\dfrac{t \in S_l \qquad \dfrac{\dfrac{\dfrac{\dfrac{\overline{y \in S}}{y \in [D]}}{y \upharpoonright [D^{-l}] \in [D^{-l}]} \quad \dfrac{}{y \upharpoonright [D^{-l}] = t}}{t \in [D^{-l}]} \qquad \dfrac{\dfrac{\dfrac{\overline{y \in S}}{P\sigma_2[l/y.l]} \quad \dfrac{}{y \upharpoonright [D^{-l}] = t}}{P\sigma_0[l/y.l]} \quad \dfrac{\dfrac{\overline{y \in S}}{y \in [D]}}{y.l \in T}}{\exists\, z \in T \bullet P\sigma_0[l/z]}}{t \in S_r} \qquad t \in S_r}{t \in S_r}\ (S^-_\exists)$$

and:

$$\dfrac{\dfrac{t \in S_r}{\exists\, z \in T \bullet P\sigma_0[l/z]} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{t \in S_r}{t \in [D^{-l}]}}{t.l_i \in T_i} \quad \cdots}{t' \in [D]} \quad \overline{P\sigma_1}}{t' \in [D \mid P]}}{t' \upharpoonright [D^{-l}] \in S_l}\ (S^+_\exists) \quad \cdots \quad \dfrac{\dfrac{\overline{t.l_i = t.l_i} \quad \overline{[D^{-l}] \sqsubseteq [D^{-l}]}}{(t' \upharpoonright [D^{-l}]).l_i = t.l_i} \quad \cdots}{t' \upharpoonright [D^{-l}] = t}}{t \in S_l}}{t \in S_l}$$

∎

As we remarked earlier, what we call existential schema hiding is often written using an explicit hiding notation: $S \setminus (l \in T)$. In this case the equation is expressed alternatively as:

$$[D \mid P] \setminus (l \in T) = [D] - [l \in T] \mid \exists\, z \in T \bullet P[l/z]].$$

Although it is senseless to consider the quantifier as *binding* the constant l, the equation reveals that it effectively *removes the constant entirely*. In other words the quantifier *hides* the constant.

For each of these schema operations we have in addition a number of easily proved *congruence rules for equality*.

PROPOSITION 4.12
The following rules are all derivable:

$$\frac{S_0 = S_1}{\neg S_0 = \neg S_1} \qquad \frac{S_0 = S_1}{S_0 \vee S_2 = S_1 \vee S_2} \qquad \frac{S_1 = S_2}{S_0 \vee S_1 = S_0 \vee S_2}$$

$$\frac{S_0 = S_1}{\exists\, l \in T \bullet S_0 = \exists\, l \in T \bullet S_1}$$

PROOF. We illustrate with the first of these. In one direction we have:

$$\dfrac{\dfrac{\dfrac{t \in \neg S_0}{t \notin S_0} \qquad \dfrac{\dfrac{S_0 = S_1}{\forall\, z \in T \bullet z \in S_0 \Leftrightarrow z \in S_1} \quad \dfrac{}{t^T \in T}}{t \in S_0 \Leftrightarrow t \in S_1}}{t \notin S_0 \Rightarrow t \notin S_1}}{\dfrac{t \notin S_1}{t \in \neg S_1}}$$
$$t \in \neg S_0 \Rightarrow t \in \neg S_1$$

In the other direction, a similar derivation provides us with $t \in \neg S_1 \Rightarrow t \in \neg S_0$. Whence:

$$\dfrac{\dfrac{\dfrac{\vdots}{t \in \neg S_0 \Rightarrow t \in \neg S_1} \quad \dfrac{\vdots}{t \in \neg S_1 \Rightarrow t \in \neg S_0}}{t \in \neg S_0 \Leftrightarrow t \in \neg S_1}}{\forall\, z \in T \bullet z \in \neg S_0 \Leftrightarrow z \in \neg S_1}$$
$$\neg S_0 = \neg S_1$$

∎

5 Logical extensions to the schema calculus

With the basic schema calculus in place, and in the context of classical logic, it is possible to extend the schema calculus with new logical operators in a straightforward manner. We now

introduce schema conjunction, schema implication and schema universal hiding:

$$S \ ::= \ \cdots \mid \ S \wedge S \ \mid \ S \Rightarrow S \ \mid \ \forall\, l \in T \bullet S.$$

The interpretations are unexceptional.

DEFINITION 5.1

$$\begin{array}{lll} (i) & [\![S_0 \wedge S_1]\!] & =_{df} \ \neg(\neg\,[\![S_0]\!] \vee \neg\,[\![S_1]\!]) \\ (ii) & [\![S_0 \Rightarrow S_1]\!] & =_{df} \ \neg\,[\![S_0]\!] \vee [\![S_1]\!] \\ (iii) & [\![\forall\, l \in T \bullet S]\!] & =_{df} \ \neg(\exists\, l \in T \bullet \neg\,[\![S]\!]). \end{array}$$

Schema conjunction enables us, as usual, to define Δ-schemas as a notational variant:

$$\Delta S =_{df} \ S \wedge S'$$

PROPOSITION 5.2
The following rules are sound for the interpretation of conjunction schemas:

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

PROOF. Consider the following derivations:

$$\dfrac{\dfrac{}{t \in \neg S_0 \vee \neg S_1} \quad \dfrac{t \upharpoonright T_0 \in S_0 \quad \dfrac{\overline{t \upharpoonright T_0 \in \neg S_0}}{t \upharpoonright T_0 \notin S_0}\ (\neg S^-)}{\bot} \quad \dfrac{t \upharpoonright T_1 \in S_1 \quad \dfrac{\overline{t \upharpoonright T_1 \in \neg S_1}}{t \upharpoonright T_1 \notin S_1}\ (\neg S^-)}{\bot}}{\dfrac{\bot}{t \notin \neg S_0 \vee \neg S_1}\ (S^-_\vee)}}{t \in \neg(\neg S_0 \vee \neg S_1)}$$

and:

$$\dfrac{\dfrac{t \in \neg(\neg S_0 \vee \neg S_1)}{t \notin \neg S_0 \vee \neg S_1}\ (\neg S^-) \quad \dfrac{\overline{t \upharpoonright T_0 \in \neg S_0}}{t \in \neg S_0 \vee \neg S_1}\ (S^+_{\vee_0})}{\bot}}{t \upharpoonright T_0 \notin \neg S_0}}{t \upharpoonright T_0 \in S_0}$$

The other elimination rule is similar. ∎

PROPOSITION 5.3
The following rules are sound for the interpretation of implication schemas:

$$\frac{t \mathrel{\dot\in} S_0 \vdash t \mathrel{\dot\in} S_1}{t \in S_0 \Rightarrow S_1} \qquad \frac{t \in S_0 \Rightarrow S_1 \quad t \mathrel{\dot\in} S_0}{t \mathrel{\dot\in} S_1}$$

PROOF. $(S^-_\vee)$, $(S^+_{\vee_0})$, $(S^+_{\vee_1})$, (S_{LEM}) and $(S^-_\vee)$, (S_{CON}). ∎

PROPOSITION 5.4
The following rules are sound for the interpretation of universally quantified schemas:

$$\frac{t(l \Rrightarrow x^{T_b}) \in S^{\mathbb{P}\, T_a}}{t \mathrel{\dot\in} \forall\, l \in T_b \bullet S}\ (S^+_\forall) \qquad \frac{t_0(l \Rrightarrow x) \mathrel{\dot\in} \forall\, l \in T_b \bullet S^{\mathbb{P}\, T_a}}{t_0(l \Rrightarrow t_1) \in S}\ (S^-_\forall)$$

where $T =_{df} T_a - [l \in T_b]$ and where, in the introduction rule, x is not free in S, any (other) subterm of t nor any assumption in the context.

PROOF. $(S^-_\exists)$, $(\forall^-)$, $(S^-_\neg)$ and $(S^+_\exists)$, $(S^+_\neg)$. ∎

We are now in a position to extend the equational logic to cover these new operators.

PROPOSITION 5.5

$$[D_0 \mid P_0] \wedge [D_1 \mid P_1] = [D_0] \sqcup [D_1] \mid P_0 \wedge P_1].$$

PROOF. This is straightforward in the presence of the equational logic for disjunction and negation schemas: $\neg(\neg[D_0 \mid P_0] \vee \neg[D_1 \mid P_1]) = \neg([D_0 \mid \neg P_0] \vee [D_1 \mid \neg P_1]) = \neg[D_0] \sqcup [D_1] \mid \neg P_0 \vee \neg P_1] = [D_0] \sqcup [D_1] \mid \neg(\neg P_0 \vee \neg P_1)] = [D_0] \sqcup [D_1] \mid P_0 \wedge P_1]$. ∎

PROPOSITION 5.6

$$[D_0 \mid P_0] \Rightarrow [D_1 \mid P_1] = [D_0] \sqcup [D_1] \mid P_0 \Rightarrow P_1].$$

PROOF. Similar to Proposition 5.5. ∎

PROPOSITION 5.7

$$\forall\, l \in T \bullet [D \mid P] = [D] - [l \in T] \mid \forall\, z \in T \bullet P[l/z]]$$

where z is fresh.

PROOF. Similar to Proposition 5.5. ∎

This last equation establishes the universal quantifier as another mechanism in the schema calculus by which hiding of labels may be effected. Usually this schema operation is called *universal quantification* whereas the dual operation is often known as *hiding*. It might be more appropriate, in view of our logical analysis, to call them *universal hiding* and *existential hiding* respectively, thereby preserving the duality in the terminology.

As expected, there are equality congruence rules for these operators. We will not state or prove them, and when we introduce further operations below we will not even trouble to mention them.

6 Schema level restriction and quantification

In this section we further extend our syntax of schemas to include schema level restriction and existential hiding at the schema level:

$$S ::= \cdots \mid S_0 \upharpoonright S_1 \mid \exists\, S_0 \bullet S_1.$$

We can also write expressions of the form $\exists\, S_0 \bullet S_1$ as $S_1 \setminus S_0$.

In order to define the first of these, schema restriction, we begin by extending filtering from terms to sets.

DEFINITION 6.1
Let $T_0 \sqsubseteq T_1$. Let z be a fresh variable.

$$C^{\mathbb{P}\, T_1} \upharpoonright \mathbb{P}\, T_0 =_{df} \{z \in T_0 \mid \exists\, x \in T_1 \bullet x \in C \wedge z = x \upharpoonright T_0\}.$$

LEMMA 6.2
The following rules are then derivable:

$$\frac{t \in C^{\mathbb{P}\, T_1} \quad T_0 \sqsubseteq T_1}{t \mathrel{\dot\in} C \upharpoonright T_0}\ (C^+_\upharpoonright) \qquad \frac{t \in C \upharpoonright T \quad x \in C, x \doteq t \vdash P}{P}\ (C^-_\upharpoonright)$$

The usual sideconditions apply to the eigenvariable x.

PROOF. These follow immediately from the rules for comprehensions and the existential quantifier. ∎

We now define schema restriction as follows:

DEFINITION 6.3

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

PROPOSITION 6.4
The following rules are sound for the interpretation of schema restriction:

$$\frac{t \in S_0 \quad t \mathrel{\dot\in} S_1}{t \mathrel{\dot\in} S_0 \upharpoonright S_1}\ (S^+_\upharpoonright) \qquad \frac{t \in S_0 \upharpoonright S_1^{\mathbb{P}\, T} \quad x \in S_0, x \mathrel{\dot\in} S_1, x \doteq t \vdash P}{P}\ (S^-_\upharpoonright)$$

The usual sideconditions apply to the eigenvariable x.

PROOF. $(S^+_\wedge)$, $(C^+_\upharpoonright)$ and $(S^-_{\wedge_0})$, $(S^-_{\wedge_1})$, $(C^-_\upharpoonright)$. ∎

Next we turn to existential hiding at the schema level.

DEFINITION 6.5
Let $T_0 \sqsubseteq T_1$.

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

PROPOSITION 6.6
The following rules are sound for the interpretation of schema level existential hiding:

$$\frac{t \in S_1^{\mathbb{P}\, T_1} \quad t \mathrel{\dot\in} S_0^{\mathbb{P}\, T_0} \quad T_0 \sqsubseteq T_1}{t \mathrel{\dot\in} \exists\, S_0 \bullet S_1}\ (\exists\, S^+)$$

$$\frac{t \in \exists\, S_0 \bullet S_1 \quad x \in S_1, x \mathrel{\dot\in} S_0, x \doteq t \vdash P}{P}\ (\exists\, S^-)$$

The usual sideconditions apply to the eigenvariable x.

PROOF. $(S^+_{\wedge})$, $(C^+_{\restriction})$ and $(S^-_{\wedge_o})$, $(S^-_{\wedge_1})$, $(C^-_{\restriction})$. ∎

Now we can introduce a notation for schema level universal quantification.

$$S \ ::= \ \cdots \forall S_0 \bullet S_1$$

with the following interpretation:

DEFINITION 6.7
Let $T_0 \sqsubseteq T_1$.

$$\left[\!\left[\forall S_0^{\mathbb{P}\, T_0} \bullet S_1^{\mathbb{P}\, T_1} \right]\!\right] =_{df} \neg \exists \left[\!\left[S_0 \right]\!\right] \bullet \neg \left[\!\left[S_1 \right]\!\right].$$

PROPOSITION 6.8
The following rules are sound for the interpretation of schema level universal quantification:

$$\frac{\cdots x_j \in T_j^m \cdots, \langle\!| \cdots m_j \Rrightarrow x_j \cdots |\!\rangle^{T_0} \in S_0^{\mathbb{P}\, T_0} \vdash t(\cdots m_j \Rrightarrow x_j \cdots)^{T_1} \in S_1^{\mathbb{P}\, T_1}}{t \mathrel{\dot\in} \forall S_0 \bullet S_1}$$

for fresh variables $\cdots x_j \cdots$.

$$\frac{t(\cdots m_j \Rrightarrow x_j \cdots) \mathrel{\dot\in} \forall S_0 \bullet S_1^{\mathbb{P}\, T_1} \quad t_0^{T_0} \in S_0}{t(\cdots m_j \Rrightarrow t_0.m_j \cdots) \in S_1}$$

PROOF. $(\exists S^-)$, (S_{CON}) and $(\exists S^+)$, (S_{CON}). ∎

There are equational laws for these three schema calculus operations.

PROPOSITION 6.9
Let $[D_1] \sqsubseteq [D_0]$.

$$[D_0 \mid P_0] \upharpoonright [D_1 \mid P_1] = [D_1 \mid (\exists([D_0] - [D_1])\sigma \bullet P_0\sigma) \wedge P_1]$$

where σ is the substitution $[\cdots l_i / z_i \cdots]$ for fresh z_i and where $\alpha([D_0] - [D_1]) = \{\cdots l_i \cdots\}$.

PROOF. $(\subseteq)$: $(S^-_{\restriction})$, $(\exists^+)$. $(\supseteq)$: $(S^+_{\restriction})$, $(\exists^-)$. ∎

PROPOSITION 6.10
Let $[D_0] \sqsubseteq [D_1]$, where $\alpha[D_0] = \{\cdots m_i \cdots\}$ and $\sigma = [\cdots m_i \cdots / \cdots z_i \cdots]$ where the z_i are fresh variables.

$$\exists[D_0 \mid P_0] \bullet [D_1 \mid P_1] = [[D_1] - [D_0] \mid \exists\, D_0\sigma \bullet (P_0 \wedge P_1)\sigma].$$

PROOF. $(\subseteq)$: $(\exists S^-)$, $(\exists^+)$. $(\supseteq)$: $(\exists S^+)$, $(\exists^-)$. ∎

PROPOSITION 6.11
Let $[D_0] \sqsubseteq [D_1]$, where $\alpha[D_0] = \{\cdots l_i \cdots\}$ and σ be the substitution $[\cdots l_i \cdots / \cdots z_i \cdots]$ where the z_i are fresh variables.

$$\forall[D_0 \mid P_0] \bullet [D_1 \mid P_1] = [[D_1] - [D_0] \mid \forall\, D_0\sigma \bullet (P_1 \Rightarrow P_0)\sigma].$$

PROOF. $\forall[D_0 \mid P_0] \bullet [D_1 \mid P_1] = \neg \exists[D_0 \mid P_0] \bullet \neg[D_1 \mid P_1] = \neg \exists[D_0 \mid P_0] \bullet [D_1 \mid \neg P_1] = \neg[D_0] - [D_1] \mid \exists\, D_1\sigma \bullet (P_0 \wedge \neg P_1)\sigma] = [D_0] - [D_1] \mid \neg \exists\, D_1\sigma \bullet (P_0 \wedge \neg P_1)\sigma] = [D_0] - [D_1] \mid \neg \exists\, D_1\sigma \bullet \neg(\neg P_0 \vee P_1)\sigma] = [D_0] - [D_1] \mid \forall\, D_1\sigma \bullet (P_0 \Rightarrow P_1)\sigma]$ ∎

7 Schema calculus for operations

In this section we examine three schema operators which are designed to apply to operation schemas. Schema composition and piping provide mechanisms for structuring operation schemas. They are similar in the sense that they both involve connections via complementary labels. In the case of composition these are the before and after states, and in the case of piping they are the inputs and outputs. Schema precondition offers a means by which an operation schema induces a state schema: its domain of definition.

Our next extension to our language of schemas is, therefore:

$$S \ ::= \ \cdots \mid S_0 \fatsemi S_1 \mid S_0 \gg S_1 \mid \mathrm{pre}\, S.$$

Our logic enables us to give elegant definitions of these new operations. In earlier work [8] we gave an account based on [4] and [18] that leads to a considerably more cumbersome analysis.

There are a number of new notions which greatly simplify the presentation. First, we treat the diacriticals of priming, shrieking and querying as bijective *operations* on labels (and *mutatis mutandis* on bindings and schema types). Hence $l^{??} = l$ and so on.

Secondly, we introduce a generalised notion of restricted equality.

DEFINITION 7.1
Let $T \sqsubseteq T_0$ and $T \sqsubseteq T_1$.

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

PROPOSITION 7.2
Let $T_0 \sqsubseteq T_1$.

$$t_0^{T_0} \doteq t_1^{T_1} \Leftrightarrow t_0 =_{T_0} t_1.$$

■

7.1 Schema composition

In what follows we shall assume that an operation schema U_i has the type: $T_i =_{df} T \sqcup I_i^? \sqcup T' \sqcup O_i^!$. This decomposition isolates the before state type T, the after state type T', the type of inputs $I_i^?$ and the type of outputs $O_i^!$. We will also write T^* for the type $T \sqcup I_0^? \sqcup I_1^? \sqcup T' \sqcup O_0^! \sqcup O_1^!$, V_0 for $T \sqcup I_0^? \sqcup O_0^!$ and V_1 for $T' \sqcup I_1^? \sqcup O_1^!$.

DEFINITION 7.3
Let z be a fresh variable.

$$U_0 \fatsemi U_1 =_{df}$$
$$\{z \in T^* \mid \exists\, z_0 \in U_0, \bullet\, z_1 \in U_1 \bullet z_0 =_{V_0} z \wedge z_1 =_{V_1} z \wedge z_0' =_T z_1\}.$$

The rules are then easy to derive.

PROPOSITION 7.4
The following rules are derivable for schema composition:

$$\frac{t_0 \in U_0 \quad t_1 \in U_1 \quad t_0 =_{V_0} t \quad t_1 =_{V_1} t \quad t_0' =_T t_1}{t \in U_0 \fatsemi U_1}$$

and:

$$\frac{t \in U_0 \fatsemi U_1 \quad y_0 \in U_0, y_1 \in U_1, y_0 =_{V_0} t, y_1 =_{V_1} t, y_0' =_T y_1 \vdash P}{P}$$

The usual sideconditions apply to the eigenvariables y_0 and y_1. ∎

7.2 Schema piping

Now we turn to schema piping. In what follows we shall assume that the operation schema U_0 has the type: $W_0 \sqcup I_0^? \sqcup W_0' \sqcup O_0^! \sqcup Y^!$, and U_1 the type $W_1 \sqcup I_1^? \sqcup Y^? \sqcup W_1' \sqcup O_1^!$ where the type Y expresses the overlap between the output labels of U_0 and the inputs of U_1.

We will also write T^* for the type $W_0 \sqcup W_1 \sqcup W_0' \sqcup W_1' \sqcup I_0^? \sqcup I_1^? \sqcup O_0^! \sqcup O_1^!$, V_0 for $W_0 \sqcup W_0' \sqcup I_0^? \sqcup O_0^!$ and V_1 for $W_1 \sqcup W_1' \sqcup O_1^! \sqcup I_1^?$.

DEFINITION 7.5
Let z be a fresh variable.

$$U_0 \gg U_1 =_{df}$$
$$\{z \in T^* \mid \exists\, z_0 \in U_0 \bullet z_1 \in U_1 \bullet z_0 =_{V_0} z \wedge z_1 =_{V_1} z \wedge z_0^! =_Y z_1^?\}.$$

The rules are then easy to derive.

PROPOSITION 7.6
The following rules are derivable for schema:

$$\frac{t_0 \in U_0 \quad t_1 \in U_1 \quad t_0 =_{V_0} t \quad t_1 =_{V_1} t \quad t_0^! =_Y t_1^?}{t \in U_0 \gg U_1}$$

and:

$$\frac{t \in U_0 \gg U_1 \quad y_0 \in U_0, y_1 \in U_1, y_0 =_{V_0} t, y_1 =_{V_1} t, y_0^! =_Y y_1^? \vdash P}{P}$$

The usual sideconditions apply to the eigenvariables y_0 and y_1. ∎

This version of piping is the most general formulation, capturing the informal description given in [14] (p. 78). Often the state spaces of the two component schemas will be identical, and the two schemas will match along *all* the inputs and outputs. However, all the complexity here is carried by the types concerned, so the rules are not more complicated in this most general case.

7.3 Schema precondition

Again, our logic leads to a great simplification. We begin by defining the notion of precondition as a *predicate* on bindings rather than directly as a state schema.

DEFINITION 7.7
Let $U^{\mathbb{P}(T^{in} \sqcup T^{out})}$ be an operation schema where T^{in} is the type of the input state and inputs, and I^{out} is the type of the output state and outputs.

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

As expected, x is in the precondition of U when there is an output for which it is the input.

PROPOSITION 7.8
The following rules are derivable for preconditions:

$$\frac{z \in U \quad z \doteq x^{T^{in}}}{Pre\ U\ x} \qquad \frac{Pre\ U\ x \quad y \in U, y \doteq x \vdash P}{P}$$

The usual sideconditions apply to the eigenvariable y. ∎

Given this we can easily construct the conventional notion of a precondition *schema*:

DEFINITION 7.9
Let z be a fresh variable.

$$\text{pre}\ U =_{df} \{z \in T^{in} \mid Pre\ U\ z\}$$

This leads immediately to the following rules.

PROPOSITION 7.10
The following rules are derivable for operation schema precondition:

$$\frac{z \in U \quad z \doteq x^{T^{in}}}{x \in \text{pre}\ U} \qquad \frac{x \in \text{pre}\ U \quad y \in U, y \doteq x \vdash P}{P}$$

The usual sideconditions apply to the eigenvariable y. ∎

8 Schema inclusion

There is very little to say about schema inclusion. This becomes a notion which is easily subsumed by the existing analysis. Recall that the basic notion of a schema is given by:

$$S ::= [D] \mid [S \mid P]$$

where we would normally write $[[D] \mid P]$ as $[D \mid P]$. Given schema conjunction we can simply take the following simple generalization:

$$S ::= \cdots \mid [\cdots, S_i, \cdots \mid P].$$

This is understood as follows:

$$[\![[\cdots, S_i, \cdots \mid P]]\!] =_{df} [\![[\cdots \wedge S_i \wedge \cdots \mid P]]\!].$$

9 Relating Z_C to the standard logic of Z

The standard approach to the logic for Z is contained in the two most recent versions of the draft standard [11, 12] and an associated technical report [9], the last of these being the most complete and mature. There are two features of the standard approach which are rather striking. First, schema components are treated as a species of variable with unusual status, hovering between the conventional notions of free and bound variable. Secondly, there is no meta-linguistic notion of substitution. Since this fact has a huge influence on the standard accounts of Z we shall devote the next section of the paper to it.

9.1 The 'frogspawn' substitutions

In place of the usual meta-linguistic notion of substitution that appears in most formal theories, in the standard accounts of Z there are two object language substitutions, both indicating the substitution of a binding: one into propositions and the other into expressions. These are written, respectively:

$$\begin{array}{ll}(i) & b \odot P \\ (ii) & b \odot e.\end{array}$$

In [9] there is a guarded suggestion that these can be defined in terms of existential quantification and definite description. Specifically:[5]

$$\begin{array}{lllll}(i) & b \odot P & =_{df} & \exists \{b\} \bullet P \\ (ii) & b \odot e & =_{df} & \mu \{b\} \bullet e.\end{array}$$

This suggestion cannot be made good within the framework of the standard logic for two related reasons. First, the rules for the existential quantifier and definite description make direct use of the frogspawn operators, so the putative definitions are proof-theoretically circular. For example:[6]

$$\frac{\Gamma \vdash b \odot P \quad \Gamma \vdash b \in S}{\Gamma \vdash \exists S \bullet P} \quad (ExistsI).$$

Secondly, there is no prospect of eliminating the use of these operators in the these rules because the standard logic for Z provides no alternative, meta-linguistic, notion of substitution with which the frogspawn might be replaced.

Given this, and having little alternative, it seems, the standard accounts characterize the frogspawn operators by a very large number of axioms and a small number of rules. There has been little or no discussion in the literature which explains why the logic should have been developed in this unusual way.

The approach seems to date back to [17] in which the prefix notation for substitution was introduced, along with the idea that substitutions could be represented by bindings. However, it is not obvious that the authors, at that time at least, were advocating an object level notion of substitution.[7] Matters are not helped by the fact that the rules and axioms for substitution in the current standard accounts are enormously redundant: many of the axioms can be derived from the rules, and *vice versa*. We illustrate this with one example.

PROPOSITION 9.1
The following axiom is derivable from the remainder of the logic:

$$\Gamma \vdash \langle\!| \; x \Rrightarrow u \; |\!\rangle \odot (e_1 = e_2) \Leftrightarrow (\langle\!| \; x \Rrightarrow u \; |\!\rangle \odot e_1 = e_2)$$

(where $\Gamma \vdash x \notin \phi e_2$).[8]

[5] In fact these are expressed as *rewrites* in the technical report. As a consequence they do not establish a definition at all. We assume that what follows is what was intended.

[6] All references to rules and axioms refer to [9].

[7] The meta-level prefix notation is, arguably, extremely elegant. Had it not been for our desire to examine the current formulation of *object*-level substitution, we might have written $z.t^T$ (or even $z \odot t^T$) in preference to the less elegant $t^T[\alpha T / z.\alpha T]$.

[8] The notation (from [9]) ϕe stands for the set of free-variables in the expression e. Note that the obvious extension to bindings of the alphabet operator α we have been using is also used later.

PROOF. We obtain, from $\Gamma \vdash \langle\!| \ x \Rrightarrow u \ |\!\rangle \odot (e_1 = e_2)$, using rule ($UseBind$), $\Gamma, AX \ \langle\!| \ x \Rrightarrow u \ |\!\rangle \ END \vdash e_1 = e_2$. Then we have: $\Gamma \vdash (\langle\!| \ x \Rrightarrow u \ |\!\rangle \odot e_1) = e_2$ using rule ($EquBind$) providing that $x \notin \phi e_2$. This argument reverses, using rules ($EquBind'$) and ($UseBind$). ∎

We can also show that the rules ($EquBind$) and ($EquBind'$) are derivable from the axioms. First, however, we need a generalization of the axiom derived in the proposition above.

LEMMA 9.2
The following generalized axiom is derivable:

$$\Gamma \vdash b \odot (e_1 = e_2) \Leftrightarrow (b \odot e_1) = e_2$$

(where $\Gamma \vdash \alpha b \cap \phi e_2 = \{\}$).

PROOF. The result follows immediately by induction on the length of the binding b from the axiom we derived in Proposition 9.1 above, and the axiom:

$$\Gamma \vdash \langle\!| \ b, x \Rrightarrow e \ |\!\rangle \odot P \Leftrightarrow b \odot (\langle\!| \ x \Rrightarrow e \ |\!\rangle \odot P).$$

∎

We then have:

PROPOSITION 9.3
The rules ($EquBind$) and ($EquBind'$) are derivable from the remainder of the logic:

$$\frac{\Gamma \ AX \ \{b\} \ END \vdash u = e \quad \Gamma \vdash \alpha b = \psi_1 \quad \Gamma \vdash \phi u = \psi_2}{\Gamma \vdash u = b \odot e} \quad (\psi_1 \cap \psi_2 = \{\})$$

$$\frac{\Gamma \vdash u = b \odot e \quad \Gamma \vdash \alpha b = \psi_1 \quad \Gamma \vdash \phi u = \psi_2}{\Gamma \ AX \ \{b\} \ END \vdash u = e} \quad (\psi_1 \cap \psi_2 = \{\}).$$

PROOF. Suppose that: $\Gamma \ AX \ \{b\} \ END \vdash u = e$ then, by rule ($UseBind$), we have: $\Gamma \vdash b \odot (u = e)$. By Lemma 9.2 we obtain: $\Gamma \vdash u = b \odot e$. This establishes rule ($EquBind$); its side-conditions arise as a consequence of the appeal to Lemma 9.2. The argument reverses, thus deriving the rule ($EquBind'$).[9] ∎

The fact that such redundancies exist (and there are very many more) is surprising enough. However, as we shall show below, it turns out that the frogspawn constructions can be shown to follow from much simpler and more straightforward ideas. There are, in fact, two approaches which we can explore. The first employs the *binding restriction operation* which we have introduced in our earlier work [6]; the second builds on the suggestion mentioned above, i.e. the use of existential quantification.

[9]Note that Proposition 9.1 generalizes to an arbitrary binding b for which $\alpha b \cap \phi e_2 = \{\}$. Consequently, we have established that the (generalized) axiom is, in the context of the remainder of the logic, *equivalent* to the rules ($EquBind$) and ($EquBind'$).

9.2 The relationship between $\odot$ and restricted membership

DEFINITION 9.4

$$t^{T_0} \odot C^{\mathbb{P}\, T_1} =_{df} t \upharpoonright T_1 \in C \qquad (T_1 \sqsubseteq T_0).$$

We then have:

PROPOSITION 9.5
The following rule is derivable:

$$\frac{\Gamma \vdash \neg\, b \odot S}{\Gamma \vdash b \odot \neg\, S}$$

PROOF. Note that $\Gamma \vdash b \odot \neg\, S$ is $\Gamma \vdash b \upharpoonright T_1 \in \neg\, S$ and $\Gamma \vdash \neg\, b \odot S$ is $\Gamma \vdash \neg\, (b \upharpoonright T_1 \in S)$. Then consider the following derivation in Z_C:

$$\dfrac{\dfrac{\Gamma \vdash \neg\, b \odot S}{\Gamma \vdash b \upharpoonright T_1 \notin S}}{\Gamma \vdash b \odot \neg\, S}\ (\neg\, S^+)$$

∎

Note that in the standard accounts there is no restriction on the types of b and S, whereas for us the restriction associated with the definition of frogspawn carries through.

Here is another example of how frogspawn interacts with the schema calculus.

PROPOSITION 9.6

$$b \odot (S_1 \vee S_2) \Leftrightarrow b \odot S_1 \vee b \odot S_2\text{`}.$$

PROOF. $\Rightarrow$: $(S^-_\vee)$, $(\vee^+_{\text{o}})$, $(\vee^+_{\text{1}})$. $\Leftarrow$: $(S^+_{\vee_{\text{o}}})$, $(S^+_{\vee_{\text{1}}})$, $(\vee^-)$. ∎

Strictly speaking, we have, in the two examples above, reinterpreted the standard use of frogspawn over schema *predicates* as instances of restricted membership over schemas as *sets*. For the more general application of frogspawn over predicates we introduce a *completion* process, which forms a (schema) set from an arbitrary proposition.

DEFINITION 9.7
Let P be a predicate containing labels as terms. The *completion of* P (written P^*) is the schema $[\cdots l_i : T_i \cdots \mid P]$ where the l_i comprise all labels occurring (at their corresponding types T_i) in P. Then we further define:

$$b \odot P =_{df} b \odot P^*.$$

The overloading of the frogspawn relation introduces no notational ambiguity.

Note that, having done all this, we have a special case of the definition: $b \odot P =_{df} b \upharpoonright T \in P^* = b \in P^*$, since $b \upharpoonright T = b$.

PROPOSITION 9.8
This rule is derivable:

$$\frac{\Gamma \vdash b \in S \wedge b \odot P}{\Gamma \vdash b \in [S \mid P]}$$

PROOF.

$$\dfrac{\dfrac{\Gamma \vdash b \in S \land b \odot P}{\Gamma \vdash b \in S} \qquad \dfrac{\dfrac{\Gamma \vdash b \in S \land b \odot P}{\Gamma \vdash b \in P^*}}{\Gamma \vdash P[\alpha S / b.\alpha S]}}{\Gamma \vdash b \in [S \mid P]}$$

∎

Part of the derivation above, it turns out, is so commonly used within much of our work on this topic, and indeed makes clear exactly what $\odot$ means when applied to a predicate, that we extract it as a lemma (which we use in the sequel without comment).

LEMMA 9.9

$$\Gamma \vdash b^T \odot P \text{ if and only if } \Gamma \vdash P[\alpha T / b.\alpha T].$$

PROOF.

$$\dfrac{\Gamma \vdash b \in P^*}{\Gamma \vdash P[\alpha T / b.\alpha T]}\ (S_1^-)$$

and:

$$\dfrac{\Gamma \vdash P[\alpha T / b.\alpha T] \quad \dfrac{}{\Gamma \vdash b \in T}\ (2.3)}{\Gamma \vdash b \in P^*}\ (S^+)$$

∎

Returning to the other direction of $SchemaMem$ we have:

PROPOSITION 9.10
This rule is derivable:

$$\dfrac{\Gamma \vdash b \in [S \mid P]}{\Gamma \vdash b \in S \land b \odot P}$$

PROOF.

$$\dfrac{\dfrac{\Gamma \vdash b \in [S \mid P]}{\Gamma \vdash b \in S}\ (S_0^-) \qquad \dfrac{\dfrac{\Gamma \vdash b \in [S^{\mathbb{P}\, T} \mid P]}{\Gamma \vdash P[\alpha T / b.\alpha T]}\ (S_1^-)}{\Gamma \vdash b \odot P}}{\Gamma \vdash b \in S \land b \odot P}\ (\land^+)$$

∎

$SchBindMem$ and $SchBindMem''$ follow directly from our definition of $\odot$ and the fact that the provisos on those rules, that the alphabets of b and S are the same, mean that $b : T$ and $S : \mathbb{P}\, T$ for some type T.

Turning to rules involving quantifiers (p. 19 of [9]), consider the rule $AllE$. First we have to interpret $\forall S^T \bullet P$, and we can do this straightforwardly by using $\forall z \in S \bullet P[\alpha T / z.\alpha T]$. We then have

PROPOSITION 9.11
The following rule is derivable:

$$\frac{\Gamma \vdash \forall z \in S \bullet P \quad \Gamma \vdash b^T \in S}{\Gamma \vdash b \odot P} \ (AllE)$$

PROOF.

$$\dfrac{\dfrac{\Gamma \vdash \forall z \in S \bullet P \quad \Gamma \vdash b^T \in S}{\Gamma \vdash P[\alpha T / b.\alpha T]}\ \forall^-}{\Gamma \vdash b \odot P}$$

∎

PROPOSITION 9.12
The following rule is derivable:

$$\frac{\Gamma \vdash b \odot P \quad \Gamma \vdash b^T \in S}{\exists z \in S \bullet P[\alpha T / z.\alpha T]} \ (ExistsI)$$

PROOF.

$$\dfrac{\dfrac{\Gamma \vdash b \odot P}{\Gamma \vdash P[\alpha T / b.\alpha T]} \quad \Gamma \vdash b^T \in S}{\exists z \in S \bullet P[\alpha T / z.\alpha T]}\ \exists^+$$

∎

9.3 Following the standard approach

As we saw in Section 9.1, the suggestion in [9] that the frogspawn operators can be defined in terms of existential quantification and definite description cannot work. In Z_C, however, it is entirely possible to develop the standard logic of frogspawn by employing the suggested definitions.

To begin with we must revise some of the infrastructural development of Z_C. As we have seen before, quantification over a schema can be represented by means of

DEFINITION 9.13

$$\begin{array}{lll} (i) & \exists S^{\mathbb{P}\, T} \bullet P & =_{df} \quad \exists z \in S \bullet P[\alpha T / z.\alpha T] \\ (ii) & \mu S^{\mathbb{P}\, T} \bullet e & =_{df} \quad \mu z \in S \bullet e[\alpha T / z.\alpha T]. \end{array}$$

Recall (see Section 4.2) that we take the θ operations of Z to be notation for the explicit characteristic bindings; so these definitions suffice to handle those notions too. In fact, an occurrence of θS in P (for example) will be equal to the value of the bound variable z.

Consequently, the definitions of frogspawn in our framework amount to

DEFINITION 9.14

$$\begin{array}{lll} (i) & b^T \odot P & =_{df} \quad \exists z \in \{b\} \bullet P[\alpha T / z.\alpha T] \\ (ii) & b^T \odot e & =_{df} \quad \mu z \in \{b\} \bullet e[\alpha T / z.\alpha T]. \end{array}$$

We can immediately illustrate the definitions with an example. One of the equations given for frogspawn is the subject of the following proposition.

PROPOSITION 9.15
The following equation holds in Z_C:

$$b \odot \neg P \Leftrightarrow \neg(b \odot P).$$

PROOF. $\Rightarrow$: ($\exists^-$) twice, ($\bot^+$), ($\neg^+$). $\Leftarrow$: ($\exists^+$) twice, ($\bot^+$), ($\neg^+$). ∎

Once again, as we did in our first formulation in Lemma 9.9, we can do much better than this via the following lemma, whose proof one can detect explicitly in the full proof of Proposition 9.15.

LEMMA 9.16

$$\Gamma \vdash b^T \odot P \text{ if and only if } \Gamma \vdash P[\alpha T / b.\alpha T].$$

PROOF. Consider the following derivations:

$$\dfrac{P[\alpha T / b.\alpha T] \quad \dfrac{\overline{b = b}}{b \in \{b\}}}{\exists\, z \in \{b\} \bullet P[\alpha T / z.\alpha T]}$$

and:

$$\dfrac{\exists\, z \in \{b\} \bullet P[\alpha T / z.\alpha T] \qquad \dfrac{\dfrac{\overline{y \in \{b\}}}{y = b} \quad \overline{P[\alpha T / y.\alpha T]}}{P[\alpha T / b.\alpha T]}}{P[\alpha T / b.\alpha T]}$$

∎

Of course, we have now shown, via Lemmas 9.9 and 9.16, that our two formulations for $\odot$ on predicates are equivalent. However, to illuminate this topic from a second perspective is useful, so we will continue a little further with the development based on this second formulation.

With Lemma 9.16 in place Proposition 9.15 becomes entirely trivial, recording nothing beyond the fact that substitution commutes with negation.

Similarly the standard rules for existential introduction and universal elimination are immediately derivable in Z_C using this lemma.

PROPOSITION 9.17
The following rules are derivable:

$$\dfrac{\Gamma \vdash b \odot P \quad \Gamma \vdash b \in S}{\Gamma \vdash \exists S \bullet P} \ (ExistsI) \qquad \dfrac{\Gamma \vdash \forall S \bullet P \quad \Gamma \vdash b \in S}{\Gamma \vdash b \odot P} \ (ForallE)$$

PROOF. Consider the following derivations:

$$\dfrac{\dfrac{\Gamma \vdash \exists z \in \{b\} \bullet P[\alpha T / z.\alpha T]}{\Gamma \vdash P[\alpha T / b.\alpha T]} \quad b \in S}{\Gamma \vdash \exists z \in S \bullet P[\alpha T / z.\alpha T]} \ (\exists^+)$$

and:

$$\dfrac{\dfrac{\forall z \in S \bullet P[\alpha T / z.\alpha T] \quad b \in S}{P[\alpha T / b.\alpha T}}{\exists z \in \{b\} \bullet P[\alpha T / z.\alpha T]} \ (\forall^-)$$

∎

The standard logic contains two rules ($UseBind$) which characterize the frogspawn operator in terms of assumptions. Specifically these are:

$$\dfrac{\Gamma \vdash b \odot P}{\Gamma \ AX \ \{b\} \ END \vdash P} \qquad \dfrac{\Gamma \ AX \ \{b\} \ END \vdash P}{\Gamma \vdash b \odot P}$$

These, or a formalization of them, are derivable from our definition in Z_C. First, we must interpret the use of the binding set $\{b\}$ as an assumption. To this end we define, for any binding b of the form: $\langle\!| \cdots l_i \Rrightarrow t_i^{T_i} \cdots |\!\rangle$, the set of equalities b^* with the form: $\cdots x_i = b.l_i \cdots$ (the variables x_i must, of course, be fresh). Then we can prove the following encodings of the rules ($UseBind$).

PROPOSITION 9.18
The following rules are derivable in Z_C:

$$\dfrac{\Gamma \vdash b \odot P}{\Gamma, b^* \vdash P[\cdots l_i / x_i \cdots]} \qquad \dfrac{\Gamma, b^* \vdash P[\cdots l_i / x_i \cdots]}{\Gamma \vdash b \odot P}$$

PROOF. ($\exists^-$) and ($\exists^+$), ($\forall^+$), ($\forall^-$), ($\Rightarrow^+$), ($\Rightarrow^-$). ∎

Next we turn to the standard rules known as ($SchemaMem$). In view of Lemma 9.16 these are simple notational variations of the schema introduction and elimination rules of Z_C.

PROPOSITION 9.19
The following rules are derivable:

$$\dfrac{\Gamma \vdash b \in S \quad \Gamma \vdash b \odot P}{\Gamma \vdash b \in [S \mid P]} \qquad \dfrac{\Gamma \vdash b \in [S \mid P]}{\Gamma \vdash b \odot P}$$

PROOF. Consider the following derivations:

$$\dfrac{b \in S \qquad \dfrac{\exists z \in \{b\} \bullet P[\alpha T / z.\alpha T]}{P[\alpha T / b.\alpha T]}}{b \in [S \mid P]}$$

and:

$$\frac{\dfrac{b \in [S \mid P]}{P[\alpha T / b.\alpha T]}}{\exists\, z \in \{b\} \bullet P[\alpha T / z.\alpha T]}$$

∎

9.4 Conclusions about frogspawn

We have not shown that all the rules in [9] and [11] which mention $\odot$ are interpretable and then derivable using our definitions of $\odot$, but we have, we feel, given enough instances to show that our first interpretation serves to explain $\odot$ in a more familiar way, i.e. as a form of restricted membership in sets. The second serves to show that we can, using Z_C, treat $\odot$ via an alternative definition (suggested in [9] but not implementable in the logic given there) which allows a better understanding of this unusual formal device.

As for the substitution into expressions that appears in [9], this seems to be definable just as ordinary substitution into expressions, i.e. what we have, all along, been writing conventionally using $[\cdots x/e \cdots]$.

We have shown that an unusual and non-standard invention of the Z Standard is not actually necessary and serves only to complicate what is really a quite straightforward language.

10 Conclusions and future work

In this paper we have provided a new core logic for the specification language Z. It differs from the logic presented in the draft Z standard [12] in several respects, all to clear advantage.

We have also shown how, from the simple kernel logic Z_C, there emerges a calculus for manipulating the main, and deservedly acclaimed, innovations of Z: the notion of the schema and the associated language of schema expressions. Rather than presenting a calculus for these expressions as a syntactic extension of Z (the methodology of the standard approach), we have shown that this calculus is derivable in a well-structured hierarchy of layers. As a result, the set-theoretic model is kept very simple, and the soundness of our logic, including the entire schema calculus, is easily demonstrated.

In addition, we have also shown that the unusual use of object-level substitution, used in the standard account, is expressible, to a large extent, in our framework in terms of much simpler and more familiar notions, and is, as a consequence, an unnecessary innovation.

What we have achieved in this paper is, in the end, very simple: by taking ordinary ZF[10] and interpreting within it just *one* new construction, that of schema types and the bindings that inhabit them, we have shown that a semantics and logic for Z can be constructed. Given the contortions that others have gone through to try to achieve the same end, this is a remarkable discovery.

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

[10] Of course very little set-theory is actually required for the analysis of Z.

Zealand; the Royal Society and the EPSRC (grant number GR/L57913) for financial assistance which has supported the development of this research. We are grateful to Rob Arthan, Jonathan Bowen, Stephen Brien, Ricardo Calderon-Cruz, John Derrick, Doug Goldson, Lindsay Groves, Thomas Santen, Susan Stepney, Bernard Sufrin, Ian Toyn, Ray Turner, Mark Utting, Sam Valentine, Norbert Völker, Jim Woodcock, and most particularly, Andrew Martin, for many useful discussions concerning Z and its formalization, and for comments on drafts of this paper.

A summary of this work was presented at ZUM'98 [6][11] and we would like to thank members of the audience for useful comments.

Finally, we also wish to thank the referees of this paper for their diligent work, and also for relating some interesting historical facts about Z to us.

References

[1] R. D. Arthan. On free type definitions in Z. In *Z User Workshop, 1991*, J. E. Nicholls, ed. pp. 40–58. Springer-Verlag, 1992.

[2] R. D. Arthan. Recursive definitions in Z. In *11th Int. Conf. ZUM'98: the Z Formal Specification Notation*, J. P. Bowen, A. Fett and M. G. Hinchey, eds. pp. 154–171. LNCS 1493, Springer-Verlag, 1998.

[3] S. M. Brien and A. P. Martin. *A Tutorial on Proof in Standard Z*. Technical monograph PRG-120, Programming Research Group, Oxford University Computing Laboratory, Oxford, UK, 1996.

[4] A. Diller. *Z: An Introduction to Formal Methods (2nd edn.)*. J. Wiley, 1994.

[5] M. C. Henson. The standard logic for Z is inconsistent. *Formal Aspects of Computing Journal*, **10**, 243–247, 1998.

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

[7] M. C. Henson and S. Reeves. Revising Z: Part I - logic and semantics. *Formal Aspects of Computing Journal*, **11**, 1999. .

[8] M. C. Henson and S. Reeves. Revising Z: Part II - logical development. *Formal Aspects of Computing Journal*, **11**, 1999. Accepted for publication.

[9] A. P. Martin. A Revised Deductive System for Z. Technical report TR98-21, Software Verification Research Centre, University of Queensland, February 1998.
Available from `http://svrc.it.uq.edu.au/~apm/logic/logictr.ps`.

[10] R. Milner. *Communication and Concurrency*. Prentice-Hall International, 1989.

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

[12] J. Nicholls, ed. *Z Notation: Version 1.3*. Z Standards Panel, 1998.

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

[14] J. M. Spivey. *The Z Notation: A Reference Manual*. Prentice Hall, 1992.

[15] J. M. Spivey. The consistency theorem for free type definitions in Z. *Formal Aspects of Computing*, **8**, 369–376, 1996.

[16] I. Toyn, ed. *Z Notation: Version 1.4*. Z Standards Panel, 1998.

[17] J. Woodcock and S. Brien. $\mathcal{W}$: A logic for Z. In *Z User Workshop, York 1991*, J. E. Nicholls, ed. pp. 77–96. Springer Verlag, 1992.

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

Received 12 October 1998

[11] Although the *published* proceedings contain a version of the schema calculus in the spirit of [7, 8] rather than that presented in this paper.