Results on Formal Stepwise Design in Z

Moshe Deutsch[1], Martin C. Henson[1], Steve Reeves[2]
[1]*Department of Computer Science, University of Essex, UK.*
[2]*Department of Computer Science, University of Waikato, New Zealand.*
{*mdeuts, hensm*}*@essex.ac.uk, stever@cs.waikato.ac.nz*

Abstract

Stepwise design involves the process of deriving a concrete model of a software system from a given abstract one. This process is sometimes known as refinement.

There are numerous refinement theories proposed in the literature, each of which stipulates the nature of the relationship between an abstract specification and its concrete counterpart.

This paper considers six refinement theories in Z that have been proposed by various people over the years. However, no systematic investigation of these theories, or results on the relationships between them, have been presented or published before. This paper shows that these theories fall into two important categories and proves that the theories in each category are equivalent.

1. Introduction

Stepwise design constitutes the underlying notion of formal program development, given an abstract formal specification to start with (see [10]). It has various different names in the Software Engineering literature, one of which is the *transformational software process model*, in which design decisions are, gradually, incorporated into the initial abstract mathematical specification of the system in question, deriving a more concrete specification at each level (see [13]). This process is also formally known as *refinement*.

In this paper, we provide a mathematical analysis of total correctness operation refinement for partial relation semantics. An important example of such a semantics is that of the specification language Z. We will examine refinement when preconditions are interpreted as minimal conditions for establishing the postconditions (they may be weakened) or fixed conditions (they are firing or trigger conditions). The former approach is covered in sections 3 and 4, and the latter in sections 5 and 6.

Our aim is to understand better the various techniques which have been proposed and to link them carefully to what appear to be, *prima facie*, alternative approaches. In particular, we will look at the standard relational completion approaches (see, for example, [17] and [3]) and relate them to a variety of proof theoretic approaches and to frameworks in which specifications are interpreted to be sets of implementations (rather in the spirit of the uses to which Martin-Löf's theory has been put [12], though our investigation takes place in classical logic).

Such an investigation becomes possible in virtue of the logic for Z reported in, for example, [9] and a novel and simple technique of rendering all the theories of refinement in a proof-theoretic form: sets of introduction and elimination rules. This leads to a uniform and simple method for proving the various equivalence results. As such, it contrasts with the more semantic based techniques employed in [2].

Our paper concludes with a review of what has been established and an agenda for further investigation.

2. Preliminaries

In this first section, we will revise a little Z logic, settling some notational conventions in the process. Additional details can be found in appendix A; however, the reader may need to consult [9] and [4] in order to fully understand the notational and meta-notational conventions.

2.1. Schemas

In [9], Z schemas, and operation schemas in particular, were formalised as sets of bindings. This captures the informal account to be found in the literature (*e.g.* [6], [17]). In this paper, we will use the meta-variable *U* (with decorations) to range over operation schemas. As an example, consider the operation schema (written horizontally):

Definition 1. $Ex_0 \mathrel{\widehat{=}} [\, \mathrm{x}, \mathrm{x}' : \mathbb{N} \mid \mathrm{x} = 0 \land \mathrm{x}' < 10 \,]$

Ex_0 has the the type $\mathbb{P}[\mathrm{x} : \mathbb{N}, \mathrm{x}' : \mathbb{N}]$, and is understood to be a set of bindings of schema type $[\mathrm{x} : \mathbb{N}, \mathrm{x}' : \mathbb{N}]$. The bind-

Proceedings of the Ninth Asia-Pacific Software Engineering Conference (APSEC'02)
0-7695-1850-8/02 $17.00 © 2002 IEEE

ings $\langle\!|\, x \Rrightarrow 0, x' \Rrightarrow n \,|\!\rangle$, where $n < 10$, are all elements of Ex_0. In fact, there are no other elements in this case. Recall that unprimed labels (such as $\mathtt{x}$) are understood to be observations of the state before the operation takes place, whereas primed labels (such as $\mathtt{x}'$) are observations of the state afterwards. Each operation schema U will have a type of the form $\mathbb{P}\,T$ where T is a schema type. The type T can, additionally, always be partitioned as the (compatible) union of its input (or before) type T^{in}, and its output (or after) type $T^{out'}$. That is, $T = T^{in} \curlyvee T^{out'}$. For the schema Ex_0 we have $T^{in} = [\mathtt{x} : \mathbb{N}]$ and $T^{out'} = [\mathtt{x}' : \mathbb{N}]$. In this paper, since we are only dealing with operation refinement, we can assume that all operation schemas have the type $\mathbb{P}\,T$ where $T = T^{in} \curlyvee T^{out'}$. With this in place, we can omit the type superscripts in most places in the sequel.

2.2. Preconditions

We can formalise the idea of the *preconditions* of an operation schema (domain of the relation, between before and after states, the schema denotes) to express the partiality involved.

Definition 2. *Let $T^{in} \preceq V$.*

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

Proposition 1. *Let y be a fresh variable, then the following introduction and elimination rules are immediately derivable for preconditions:*

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

□

Clearly, the precondition of Ex_0 is not (and for operation schemas in general, will not be) the whole of $[\mathtt{x} : \mathbb{N}]$ (in general, T^{in}). In this sense operation schemas denote partial relations.

2.3. The distinguished element

The concept of an additional element, sometimes called *bottom*, used in *total correctness*, is well known in the literature. It is, for example, referred to as the "undefined" element in [17] or as "nontermination" in [7]. However, neither of these systematically investigates its mathematical role in total correctness based refinement.

In this paper we will simply call it the *distinguished* element, denoted by the symbol $\perp$. We show in appendix A how $\perp$ terms are incorporated in the existing types of $\mathcal{Z}_{\mathcal{C}}$ [9], in order to obtain a *conservative extension* framework, $\mathcal{Z}_{\mathcal{C}}^{\perp}$ [4]. Furthermore, we present the notion of the *natural carrier* set for each $\mathcal{Z}_{\mathcal{C}}^{\perp}$ type, which excludes $\perp$ terms or, in case of a schema type, bindings that contain at least one observation bound to $\perp$.

Following the above insight, we provide the semantics for atomic schemas:

Definition 3. $[T \mid P] =_{df} \{z \in T \mid z.P\}$

Note that this definition draws bindings from the natural carrier of the type T. As a consequence, writing $t(\perp)$ for any term of the appropriate type, which contains an instance of the constant $\perp$, we have:

Lemma 1.

$$\frac{t(\perp) \in U}{false}$$

□

The mathematical analysis throughout the development, particularly in sections 4 and 6, provides an important insight regarding to the role of $\perp$ in chaotic and abortive total correctness based refinement.

3. Refinement with preconditions considered minimal

The partial relation semantics of operation schemas in Z raises an immediate question: what does it mean for one operation schema to refine another? More generally, we are asking: what does it mean for one partial relation to refine another?

We begin by introducing three distinct notions of refinement, based on three distinct answers to the questions above and then we go on to compare them. This serves to illuminate them all, particularly the notion based on the lifted-totalisation (see below), which is the *de facto* standard for Z.

3.1. S-refinement

In this section, we introduce a purely proof theoretic characterisation of refinement, which is closely connected to refinement as introduced by Spivey (hence "S"-refinement) in, for example, [15] and as discussed in [11] and [14].

This notion is based on two basic observations regarding the properties one expects in a refinement: firstly, that a refinement may involve the reduction of non-determinism; secondly that, if preconditions are minimal, a refinement may also involve the expansion of the domain of definition. Put another way, we have a refinement providing that *postconditions do not weaken* (we do not permit an increase in non-determinism in a refinement) and that *preconditions do* *not strengthen* (we do not permit requirements in the domain of definition to disappear in a refinement).

This notion can be captured by forcing the refinement relation to hold *exactly* when these conditions apply. S-refinement is written $U_0 \sqsupseteq_s U_1$ and is given by the definition that leads directly to the following rules:

Proposition 2. *Let z, z_0, z_1 be fresh variables.*

$$\frac{Pre\ U_1\ z \vdash Pre\ U_0\ z \qquad Pre\ U_1\ z_0, z_0 \star z_1' \in U_0 \vdash z_0 \star z_1' \in U_1}{U_0 \sqsupseteq_s U_1}$$

$$\frac{U_0 \sqsupseteq_s U_1 \qquad Pre\ U_1\ t}{Pre\ U_0\ t}\ (\sqsupseteq_{s\ 0}^{\ -})$$

$$\frac{U_0 \sqsupseteq_s U_1 \qquad Pre\ U_1\ t_0 \qquad t_0 \star t_1' \in U_0}{t_0 \star t_1' \in U_1}\ (\sqsupseteq_{s\ 1}^{\ -})$$

□

This theory does not depend on, and makes no reference to, the $\perp$ value. It can be formalised in the core theory $\mathcal{Z}_{\mathcal{C}}$.

3.2. The chaotic relational completion

In this section, we review $W_\bullet$-*refinement* (written $U_0 \sqsupseteq_{w_\bullet} U_1$): this notion, adapted from, for example, [17] (hence "W" for Woodcock), is based on a relational completion operator. For notational convenience we will write $T^\star$ for the set $T_\perp^{in} \star T_\perp^{out'}$ (note the use of $\star$ for sets, as opposed to $\curlyvee$ used for types).

The *lifted totalisation* of a set of bindings can be defined as follows:

Definition 4.

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

Proposition 3. *The following introduction and elimination rules are derivable for lifted totalised sets:*

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

and:

$$\frac{t_0 \star t_1' \in \overset{\bullet}{U} \qquad Pre\ U\ t_0}{t_0 \star t_1' \in U}\ (\bullet_0^-) \qquad \frac{t_0 \star t_1' \in \overset{\bullet}{U}}{t_0 \star t_1' \in T^\star}\ (\bullet_1^-)$$

□

Lemma 2. *The following are derivable:*

$$\frac{}{U \subseteq \overset{\bullet}{U}}\ (i) \qquad \frac{}{\perp \in \overset{\bullet}{U}}\ (ii)$$

$$\frac{\neg Pre\ U\ t_0 \qquad t_0 \in T_\perp^{in} \qquad t_1' \in T_\perp^{out'}}{t_0 \star t_1' \in \overset{\bullet}{U}}\ (iii)$$

□

Lemmas 2(i), (ii) and (iii) demonstrate that definition 4 is consistent with the intentions described in [17] chapter 16: the underlying partial relation is contained in the completion; the $\perp$ element is present in the relation, and more generally, each value outside the preconditions maps to every value in the range of the relation. Then $W_\bullet$-*refinement* is defined as follows:

Definition 5. $U_0 \sqsupseteq_{w_\bullet} U_1 =_{df} \overset{\bullet}{U_0} \subseteq \overset{\bullet}{U_1}$

Obvious introduction and elimination rules follow from this.

3.3. F-refinement

To a logician, a specification resembles a theory; so a natural question is: what are the models of the theory? A computer scientist may ask a closely related question: when is a program an implementation of the specification? We will, in this section, consider deterministic programs and model them as (total) functions.

From the logical perspective, we are interested in all the models of a theory, so given a putative model g and a theory U, we would be inclined to write:

$$g \models U$$

to represent the statement that g is a model of U. Within our application area in computer science, we might prefer to read this as a relation of *implementation*. To signal this interpretation, we shall, in fact, write this judgement as:

$$g \mathrel{\in\!\!\!\!\shortmid} U$$

to be pronounced "g implements (is an implementation of) U".

Our third approach to refinement is to consider specifications as sets of implementations and then to define refinement as containment of implementations.

Definition 6.

$$g \mathrel{\in\!\!\!\!\shortmid}_f U =_{df} (\forall z \in T_\perp^{in} \bullet Pre\ U\ z \Rightarrow z \star (g\ z)' \in U) \wedge$$
$$g \in T_\perp^{in} \rightarrow T_\perp^{out'}$$

Then we can prove the following.

Proposition 4. *Let z be a fresh variable, then the following introduction and elimination rules are derivable:*

$$\frac{z \in T_\perp^{in}, Pre\ U\ z \vdash z \star (g\ z)' \in U \qquad g \in T_\perp^{in} \rightarrow T_\perp^{out'}}{g \mathrel{\in\!\!\!\!\shortmid}_f U}\ (\mathrel{\in\!\!\!\!\shortmid}_f^+)$$

$$\frac{g \mathrel{\in\!\!\!\!\shortmid}_f U \qquad Pre\ U\ t \qquad t \in T_\perp^{in}}{t \star (g\ t)' \in U}\ (\mathrel{\in\!\!\!\!\shortmid}_{f_0}^-) \qquad \frac{g \mathrel{\in\!\!\!\!\shortmid}_f U}{g \in T_\perp^{in} \rightarrow T_\perp^{out'}}\ (\mathrel{\in\!\!\!\!\shortmid}_{f_1}^-)$$

□

This is sufficient technical development to allow us to explore refinement. We can answer the question: when is U_0 a refinement of U_1? A reasonable answer is: when any implementation of U_0 is also an implementation of U_1. After all, we wish to be able to replace any specification U_1 by its refinement U_0, and if all potential implementations of the latter are implementations of this former we are quite safe. Thus we are led to:

Definition 7. $\widehat{U} =_{df} \{z \mid z \in\!\!\!\!\!+_f U\}$

Then we have F-refinement ("F" for function).

Definition 8. $U_0 \sqsupseteq_f U_1 =_{df} \widehat{U_0} \subseteq \widehat{U_1}$

Obvious introduction and elimination rules for F-refinement follow from this definition.

4. Three equivalent theories

In this section, we demonstrate that our three theories of refinement are all equivalent. In doing this, we will see clearly the critical role that the $\perp$ value plays.

We shall be showing that all judgements of refinement in one theory are contained among the refinements sanctioned by another. Such results will always be established proof-theoretically. Specifically, we will show that the refinement relation of a theory T_0 satisfies the elimination rule (or rules) for refinement of another theory T_1. Since the elimination rules and introduction rules of a theory enjoy the usual symmetry property, this is sufficient to show that all T_0-refinements are also T_1-refinements.

4.1. W$_\bullet$-refinement and S-refinement are equivalent

We begin by showing that W$_\bullet$-refinement satisfies the two S-refinement elimination rules. Firstly the rule for preconditions.

Proposition 5. *The following rule is derivable:*

$$\dfrac{U_0 \sqsupseteq_{w_\bullet} U_1 \quad \mathit{Pre}\ U_1\ t}{\mathit{Pre}\ U_0\ t}$$

Proof.

$$\dfrac{\dfrac{\dfrac{\dfrac{U_0 \sqsupseteq_{w_\bullet} U_1 \quad \begin{matrix}\delta\\ \vdots\\ t\star \perp' \in \overset{\bullet}{U_0}\end{matrix}}{t\star \perp' \in \overset{\bullet}{U_1}} \quad \mathit{Pre}\ U_1\ t}{t\star \perp' \in U_1}}{\mathit{false}}\ (Lem.\ 1)}{\mathit{Pre}\ U_0\ t}\ (1)$$

Where δ stands for the following branch:

$$\dfrac{\dfrac{}{\neg \mathit{Pre}\ U_0\ t}\ (1) \quad \dfrac{\mathit{Pre}\ U_1\ t}{t \in T_\perp^{in}} \quad \dfrac{}{\perp' \in T_\perp^{out'}}}{t_0\star \perp' \in \overset{\bullet}{U_0}}\ (Lem.\ 2(iii))$$

□

Turning now to the second elimination rule in S-refinement.

Proposition 6. *The following rule is derivable:*

$$\dfrac{U_0 \sqsupseteq_{w_\bullet} U_1 \quad \mathit{Pre}\ U_1\ t_0 \quad t_0 \star t_1' \in U_0}{t_0 \star t_1' \in U_1}$$

Proof.

$$\dfrac{\dfrac{U_0 \sqsupseteq_{w_\bullet} U_1 \quad \dfrac{t_0 \star t_1' \in U_0}{t_0 \star t_1' \in \overset{\bullet}{U_0}}\ (Lem.\ 2(i))}{t_0 \star t_1' \in \overset{\bullet}{U_1}} \qquad \mathit{Pre}\ U_1\ t_0}{t_0 \star t_1' \in U_1}$$

□

Theorem 1. $U_0 \sqsupseteq_{w_\bullet} U_1 \Rightarrow U_0 \sqsupseteq_s U_1$

Proof. This follows immediately, by ($\sqsupseteq_s^+$), from propositions 5 and 6[1]. □

We now show that S-refinement satisfies the W$_\bullet$-elimination rule.

Proposition 7.

$$\dfrac{U_0 \sqsupseteq_s U_1 \quad t \in \overset{\bullet}{U_0}}{t \in \overset{\bullet}{U_1}}$$

Proof.

$$\dfrac{\dfrac{t \in \overset{\bullet}{U_0}}{t \in T^\star} \quad \dfrac{U_0 \sqsupseteq_s U_1 \quad \dfrac{}{\mathit{Pre}\ U_1\ t}\ (1) \quad \begin{matrix}\delta\\ \vdots\\ t \in U_0\end{matrix}}{t \in U_1}}{t \in \overset{\bullet}{U_1}}\ (1)$$

Where δ is:

$$\dfrac{t \in \overset{\bullet}{U_0} \quad \dfrac{U_0 \sqsupseteq_s U_1 \quad \dfrac{}{\mathit{Pre}\ U_1\ t}\ (1)}{\mathit{Pre}\ U_0\ t}}{t \in U_0}$$

□

[1] The proofs of such theorems are always automatic by the structural symmetry between introduction and elimination rules. We shall not give them in future.

Theorem 2. $U_0 \sqsupseteq_s U_1 \Rightarrow U_0 \sqsupseteq_{w_\bullet} U_1$ □

Theorems 1 and 2 together establish that the theories of S-refinement and W$_\bullet$-refinement are equivalent.

4.2. F-refinement and W$_\bullet$-refinement are equivalent (in $\mathcal{Z}_C^\perp$ + AC)

4.2.1. R-refinement. We begin this analysis by defining, by way of an intermediate stage, the set of total functions compatible with an operation schema. This forms a bridge between F-refinement and W$_\bullet$-refinement.

Definition 9. $\overline{U} =_{df} \{z \in T_\perp^{in} \to T_\perp^{out'} \mid z \subseteq \overset{\bullet}{U}\}$

Then we have:

Definition 10. $g \mathrel{⋲}_r U =_{df} g \in \overline{U}$

Then R-refinement is simply: $U_0 \sqsupseteq_r U_1 =_{df} \overline{U_0} \subseteq \overline{U_1}$ with the obvious introduction and elimination rules.

4.2.2. R-refinement and W$_\bullet$-refinement are equivalent. We show that R-refinement satisfies the W$_\bullet$-refinement elimination rule and that W$_\bullet$-refinement satisfies the R-refinement elimination rule.

Proposition 8. *The following rules are derivable:*

$$\dfrac{U_0 \sqsupseteq_r U_1 \quad t \in \overset{\bullet}{U}_0}{t \in \overset{\bullet}{U}_1}\,(i) \qquad \dfrac{U_0 \sqsupseteq_{w_\bullet} U_1 \quad g \in \overline{U_0}}{g \in \overline{U_1}}\,(ii)$$

Proof. For (i), the proof requires the axiom of choice (see the step labelled (AC) below).

$$\dfrac{\dfrac{t \in \overset{\bullet}{U}_0}{\exists g \in T_\perp^{in} \to T_\perp^{out'} \bullet t \in g \wedge g \subseteq \overset{\bullet}{U}_0}\,(AC) \qquad \begin{matrix}\delta \\ \vdots \\ t \in \overset{\bullet}{U}_1\end{matrix}}{t \in \overset{\bullet}{U}_1}\,(1)$$

Where δ is:

$$\dfrac{\dfrac{\dfrac{U_0 \sqsupseteq_r U_1 \qquad \dfrac{\dfrac{}{y \in T_\perp^{in} \to T_\perp^{out'}}\,(1) \quad \dfrac{}{y \subseteq \overset{\bullet}{U}_0}\,(1)}{y \in \overline{U_0}}}{y \in \overline{U_1}}}{y \subseteq \overset{\bullet}{U}_1} \qquad \dfrac{}{t \in y}\,(1)}{t \in \overset{\bullet}{U}_1}$$

For (ii), consider the following derivation:

$$\dfrac{\dfrac{g \in \overline{U_0}}{g \in T_\perp^{in} \to T_\perp^{out'}} \qquad \dfrac{\dfrac{U_0 \sqsupseteq_{w_\bullet} U_1 \qquad \dfrac{\dfrac{g \in \overline{U_0}}{g \subseteq \overset{\bullet}{U}_0} \quad \dfrac{}{t \in g}\,(1)}{t \in \overset{\bullet}{U}_0}}{t \in \overset{\bullet}{U}_1}}{g \subseteq \overset{\bullet}{U}_1}\,(1)}{g \in \overline{U_1}}$$

□

From this we immediately get implication in both directions:

Theorem 3. $U_0 \sqsupseteq_r U_1 \Leftrightarrow U_0 \sqsupseteq_{w_\bullet} U_1$ □

4.2.3. R-refinement and F-refinement are equivalent. In this case, we show that the notions of *implementation* (rather than refinement) are equivalent by the same strategy involving elimination rules. We first establish that F-implementation implies R-implementation:

Proposition 9. *The following rules are derivable:*

$$\dfrac{g \mathrel{⋲}_f U}{g \subseteq \overset{\bullet}{U}} \qquad \dfrac{g \mathrel{⋲}_f U}{g \in T_\perp^{in} \to T_\perp^{out'}}$$

Proof.

$$\dfrac{\dfrac{\dfrac{\dfrac{g \mathrel{⋲}_f U}{g \in T_\perp^{in} \to T_\perp^{out'}} \quad \dfrac{}{z_0 \star z_1' \in g}\,(1)}{z_0 \star z_1' \in T^\star} \qquad \begin{matrix}\delta_0 \\ \vdots \\ z_0 \star z_1' \in U\end{matrix}}{z_0 \star z_1' \in \overset{\bullet}{U}}\,(2)}{g \subseteq \overset{\bullet}{U}}\,(1)$$

Where δ_0 is:

$$\dfrac{\dfrac{g \mathrel{⋲}_f U \quad \dfrac{}{Pre\ U\ z_0}\,(2) \quad \dfrac{\dfrac{}{Pre\ U\ z_0}\,(2)}{z_0 \in T_\perp^{in}}}{z_0 \star (g\ z_0)' \in U} \qquad \begin{matrix}\delta_1 \\ \vdots \\ z_1' = (g\ z_0)'\end{matrix}}{z_0 \star z_1' \in U}$$

Where δ_1 is:

$$\dfrac{\dfrac{g \mathrel{⋲}_f U}{g \in T_\perp^{in} \to T_\perp^{out'}} \quad \dfrac{}{z_0 \star z_1' \in g}\,(1)}{z_1' = (g\ z_0)'}$$

The second rule is immediate. □

Theorem 4. $g \mathrel{⋲}_f U \Rightarrow g \mathrel{⋲}_r U$ □

Now we show that R-implementation implies F-implementation.

Proposition 10.

$$\frac{g \mathrel{\in\!\!\!\!+}_r U \quad Pre\ U\ t \quad t \in T_{\perp}^{in}}{t \star (g\ t)' \in U} \qquad \frac{g \mathrel{\in\!\!\!\!+}_r U}{g \in T_{\perp}^{in} \to T_{\perp}^{out'}}$$

Proof.

$$\cfrac{\cfrac{\cfrac{g \mathrel{\in\!\!\!\!+}_r U}{g \subseteq \overset{\bullet}{U}} \quad \cfrac{\cfrac{g \mathrel{\in\!\!\!\!+}_r U}{g \in T_{\perp}^{in} \to T_{\perp}^{out'}} \quad t \in T_{\perp}^{in}}{t \star (g\ t)' \in g}}{t \star (g\ t)' \in \overset{\bullet}{U}} \qquad Pre\ U\ t}{t \star (g\ t)' \in U}$$

The second rule is immediate. □

Theorem 5. $g \mathrel{\in\!\!\!\!+}_r U \Rightarrow g \mathrel{\in\!\!\!\!+}_f U$ □

Then, from theorems 4 and 5, we see that the two notions of implementation are equivalent. Hence, so are the two notions of refinement.

Despite their superficial dissimilarity, all three theories are, then, equivalent. We will examine in section 7 some consequences of these results.

5. Refinement with preconditions considered fixed

We now introduce three further notions of refinement; in this case, where non-determinism may be reduced but where the preconditions are considered fixed.

5.1. SP-refinement

This is an alternative proof theoretic characterisation of refinement, which is closely connected to refinement in the *behavioural approach*, as discussed, for example, in [3] and [16].

This special case of S-Refinement may involve reduction of non-determinism but insists on the *stability of the preconditions*. SP-refinement is written $U_0 \sqsupseteq_{sp} U_1$ and is given by the definition that leads directly to the following rules:

Proposition 11. *Let z, z_0, z_1 be fresh variables.*

$$\frac{Pre\ U_1\ z \vdash Pre\ U_0\ z \quad z_0 \star z_1' \in U_0 \vdash z_0 \star z_1' \in U_1}{U_0 \sqsupseteq_{sp} U_1}\ (\sqsupseteq_{sp}^{+})$$

$$\frac{U_0 \sqsupseteq_{sp} U_1 \quad Pre\ U_1\ t}{Pre\ U_0\ t}\ (\sqsupseteq_{sp_0}^{-})$$

$$\frac{U_0 \sqsupseteq_{sp} U_1 \quad t_0 \star t_1' \in U_0}{t_0 \star t_1' \in U_1}\ (\sqsupseteq_{sp_1}^{-})$$

□

5.2. The abortive relational completion

In this section we review $W_{\square}$-*refinement* (written $U_0 \sqsupseteq_{w_{\square}} U_1$). This notion is based on a relational completion operator, but this time takes an *abortive approach* with respect to values outside the preconditions (see, for example, [1] and [3]). The *abortive lifted totalisation* of a set of bindings is defined as follows:

Definition 11.

$$\overset{\square}{U} =_{df} \{z_0 \star z_1' \in T^{\star} \mid z_0 \star z_1' \in U \vee (\neg\ Pre\ U\ z_0 \wedge z_1' = \perp')\}$$

Proposition 12. *The following introduction and elimination rules are derivable:*

$$\frac{t_0 \star t_1' \in U}{t_0 \star t_1' \in \overset{\square}{U}}\ (\square_0^{+}) \qquad \frac{t_0 \star t_1' \in T^{\star} \quad \neg\ Pre\ U\ t_0 \quad t_1' = \perp'}{t_0 \star t_1' \in \overset{\square}{U}}\ (\square_1^{+})$$

$$\frac{t_0 \star t_1' \in \overset{\square}{U} \quad t_0 \star t_1' \in U \vdash P \quad \neg\ Pre\ U\ t_0, t_1' = \perp' \vdash P}{P}\ (\square_0^{-})$$

$$\frac{t_0 \star t_1' \in \overset{\square}{U}}{t_0 \star t_1' \in T^{\star}}\ (\square_1^{-})$$

□

Note that it is, sometimes, useful to use the following version of $(\square^{+})$ rule (e.g. in the proof of proposition 18(i)), which is based upon implication introduction, rather than disjunction introduction.

Proposition 13.

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

□

Lemma 3. *The following are derivable:*

$$\frac{}{\overset{\square}{U} \subseteq \overset{\bullet}{U}}\ (i) \qquad \frac{}{\perp \in \overset{\square}{U}}\ (ii) \qquad \frac{\neg\ Pre\ U\ t \quad t \in T_{\perp}^{in}}{t \star \perp' \in \overset{\square}{U}}\ (iii)$$

$$\frac{t_0 \star t_1' \in \overset{\square}{U} \quad t_1' \neq \perp'}{t_0 \star t_1' \in U}\ (iv) \qquad \frac{t_0 \star t_1' \in \overset{\square}{U} \quad t_0 = \perp}{t_1' = \perp'}\ (v)$$

□

$W_{\square}$-*refinement* is then defined as follows:

Definition 12. $U_0 \sqsupseteq_{w_{\square}} U_1 =_{df} \overset{\square}{U_0} \subseteq \overset{\square}{U_1}$

Obvious introduction and elimination rules follow from this.

5.3. FP-refinement

Like F-refinement, *FP-refinement* considers specifications to be sets of implementations, and then we define refinement as containment of implementations. Unlike F-refinement, implementations abort outside the domain of definition, rather than behave chaotically.

Definition 13.

$$g \Subset_{fp} U =_{df} (\forall z \in T_\perp^{in} \bullet Pre\ U\ z \vee (g\ z)' \neq \perp' \Rightarrow \\ z \star (g\ z)' \in U) \wedge g \in T_\perp^{in} \to T_\perp^{out'}$$

Then we can prove the following.

Proposition 14. *Let δ be $g \in T_\perp^{in} \to T_\perp^{out'}$ and z be a fresh variable, then the following introduction and elimination rules are derivable:*

$$\dfrac{z \in T_\perp^{in}, Pre\ U\ z \vee (g\ z)' \neq \perp' \vdash z \star (g\ z)' \in U \quad \begin{array}{c}\delta\\ \vdots\end{array}}{g \Subset_{fp} U}\ (\Subset_{fp}^{+})$$

$$\dfrac{g \Subset_{fp} U \quad t \in T_\perp^{in} \quad Pre\ U\ t}{t \star (g\ t)' \in U}\ (\Subset_{fp_0}^{-})$$

$$\dfrac{g \Subset_{fp} U \quad t \in T_\perp^{in} \quad (g\ t)' \neq \perp'}{t \star (g\ t)' \in U}\ (\Subset_{fp_1}^{-}) \qquad \dfrac{g \Subset_{fp} U}{g \in T_\perp^{in} \to T_\perp^{out'}}$$

□

Definition 14. $\overset{\boxdot}{U} =_{df} \{z \mid z \Subset_{fp} U\}$

Then we have FP-refinement.

Definition 15. $U_0 \sqsupseteq_{fp} U_1 =_{df} \overset{\boxdot}{U_0} \subseteq \overset{\boxdot}{U_1}$

Obvious introduction and elimination rules for FP-refinement follow from this definition.

6. Three equivalent theories

In this section, we demonstrate that this second set of three theories of refinement are all equivalent.

6.1. W$_\square$-refinement and SP-refinement are equivalent

We begin by showing that W$_\square$-refinement satisfies the two SP-refinement elimination rules.

Proposition 15. *The following rules are derivable:*

$$\dfrac{U_0 \sqsupseteq_{w_\square} U_1 \quad Pre\ U_1\ t}{Pre\ U_0\ t}\ (i) \qquad \dfrac{U_0 \sqsupseteq_{w_\square} U_1 \quad t_0 \star t_1' \in U_0}{t_0 \star t_1' \in U_1}\ (ii)$$

Proof. For (i), consider the following derivation:

$$\dfrac{\dfrac{\begin{array}{c}\delta_0\\ \vdots\\ t\star \perp' \in \overset{\square}{U_1}\end{array} \quad \dfrac{\dfrac{}{t\star \perp' \in U_1}\ (2)}{false} \quad \dfrac{Pre\ U_1\ t \quad \dfrac{}{\neg\ Pre\ U_1\ t}\ (2)}{false}}{false}\ (2)}{Pre\ U_0\ t}\ (1)$$

Where δ_0 stands for the following branch:

$$\dfrac{U_0 \sqsupseteq_{w_\square} U_1 \quad \dfrac{\dfrac{}{\neg\ Pre\ U_0\ t}\ (1) \quad \dfrac{Pre\ U_1\ t}{t \in T_\perp^{in}}}{t\star \perp' \in \overset{\square}{U_0}}\ (Lem.\ 3(iii))}{t\star \perp' \in \overset{\square}{U_1}}$$

For (ii), consider the following derivation:

$$\dfrac{\begin{array}{c}\delta_1\\ \vdots\\ t_0 \star t_1' \in \overset{\square}{U_1}\end{array} \quad \dfrac{}{t_0 \star t_1' \in U_1}\ (1) \quad \dfrac{\dfrac{\dfrac{t_0 \star t_1' \in U_0 \quad \dfrac{}{t_1' = \perp'}\ (1)}{t_0 \star \perp' \in U_0}}{false}}{t_0 \star t_1' \in U_1}}{t_0 \star t_1' \in U_1}\ (1)$$

Where δ_1 is:

$$\dfrac{U_0 \sqsupseteq_{w_\square} U_1 \quad \dfrac{t_0 \star t_1' \in U_0}{t_0 \star t_1' \in \overset{\square}{U_0}}}{t_0 \star t_1' \in \overset{\square}{U_1}}$$

□

Theorem 6. $U_0 \sqsupseteq_{w_\square} U_1 \Rightarrow U_0 \sqsupseteq_{sp} U_1$ □

We now show that SP-refinement satisfies the W$_\square$-elimination rule.

Proposition 16.

$$\dfrac{U_0 \sqsupseteq_{sp} U_1 \quad t_0 \star t_1' \in \overset{\square}{U_0}}{t_0 \star t_1' \in \overset{\square}{U_1}}$$

Proof.

$$\dfrac{t_0 \star t_1' \in \overset{\square}{U_0} \quad \begin{array}{c}\delta_0\\ \vdots\\ t_0 \star t_1' \in \overset{\square}{U_1}\end{array} \quad \begin{array}{c}\delta_1\\ \vdots\\ t_0 \star t_1' \in \overset{\square}{U_1}\end{array}}{t_0 \star t_1' \in \overset{\square}{U_1}}\ (1)$$

Where δ_0 stands for the following branch:

$$\dfrac{\dfrac{U_0 \sqsupseteq_{sp} U_1 \quad \overline{t_0 \star t_1' \in U_0}^{\,(1)}}{t_0 \star t_1' \in U_1}}{t_0 \star t_1' \in \overset{\Box}{U}_1}$$

and δ_1 stands for the following branch:

$$\dfrac{\dfrac{t_0 \star t_1' \in \overset{\Box}{U}_0}{t_0 \star t_1' \in T^\star} \quad \dfrac{U_0 \sqsupseteq_{sp} U_1 \quad \overline{\neg\, Pre\, U_0\, t_0}^{\,(1)}}{\neg\, Pre\, U_1\, t_0} \quad \overline{t_1' = \perp'}^{\,(1)}}{t_0 \star t_1' \in \overset{\Box}{U}_1}$$

□

Theorem 7. $U_0 \sqsupseteq_{sp} U_1 \Rightarrow U_0 \sqsupseteq_{w_\Box} U_1$ □

Theorems 6 and 7 together establish that the theories of SP-refinement and W□-refinement are equivalent.

6.2. FP-refinement and W□-refinement are equivalent (in $\mathcal{Z}_C^\perp$ + AC)

6.2.1. RP-refinement. As in section 4.2, we begin the analysis by defining the set of total functions, compatible with an operation schema, which forms a bridge between FP-refinement and W□-refinement.

Definition 16. $\widetilde{U} =_{df} \{z \in T_\perp^{in} \to T_\perp^{out'} \mid z \subseteq \overset{\Box}{U}\}$

Then we have:

Definition 17. $g \Subset_{rp} U =_{df} g \in \widetilde{U}$

Then RP-refinement is simply: $U_0 \sqsupseteq_{rp} U_1 =_{df} \widetilde{U_0} \subseteq \widetilde{U_1}$ with the usual introduction and elimination rules.

6.2.2. RP-refinement and W□-refinement are equivalent. Likewise, we show that RP-refinement satisfies the W□-refinement elimination rule and that W□-refinement satisfies the RP-refinement elimination rule.

Proposition 17. *The following rules are derivable:*

$$\dfrac{U_0 \sqsupseteq_{rp} U_1 \quad t \in \overset{\Box}{U}_0}{t \in \overset{\Box}{U}_1}\ (i) \qquad \dfrac{U_0 \sqsupseteq_{w_\Box} U_1 \quad g \in \widetilde{U_0}}{g \in \widetilde{U_1}}\ (ii)$$

Proof. The proofs are identical to the ones of proposition 8(i) and (ii) (respectively), where every $\overset{\bullet}{U}_i$ is substituted by $\overset{\Box}{U}_i$, every $\overline{U_i}$ is substituted by $\widetilde{U}_i$ ($i \in 2$) and $\sqsupseteq_r$, $\sqsupseteq_{w_\bullet}$ are respectively substituted by $\sqsupseteq_{rp}$, $\sqsupseteq_{w_\Box}$. □

From this we have:

Theorem 8. $U_0 \sqsupseteq_{rp} U_1 \Leftrightarrow U_0 \sqsupseteq_{w_\Box} U_1$ □

6.2.3. RP-refinement and FP-refinement are equivalent. We now show that the notions of implementation are equivalent by the same strategy involving elimination rules. We first establish that FP-implementation implies RP-implementation:

Proposition 18. *The following rules are derivable:*

$$\dfrac{g \Subset_{fp} U}{g \subseteq \overset{\Box}{U}}\ (i) \qquad \dfrac{g \Subset_{fp} U}{g \in T_\perp^{in} \to T_\perp^{out'}}\ (ii)$$

Proof. We do not provide the proof for (i), due to its substantial length and complexity. Notwithstanding, should the reader is interested, a complete account for that is provided in [5]. The second rule is immediate. □

Theorem 9. $g \Subset_{fp} U \Rightarrow g \Subset_{rp} U$ □

Now we show that RP-implementation implies FP-implementation.

Proposition 19. *The following rules are derivable:*

$$\dfrac{g \Subset_{rp} U \quad t \in T_\perp^{in} \quad Pre\, U\, t}{t \star (g\, t)' \in U}\ (i)$$

$$\dfrac{g \Subset_{rp} U \quad t \in T_\perp^{in} \quad (g\, t)' \neq \perp'}{t \star (g\, t)' \in U}\ (ii) \qquad \dfrac{g \Subset_{rp} U}{g \in T_\perp^{in} \to T_\perp^{out'}}\ (iii)$$

Proof. The first rule:

$$\dfrac{\dfrac{\begin{array}{c}\delta \\ \vdots \\ t \star (g\, t)' \in \overset{\Box}{U}\end{array}}{t \star (g\, t)' \in \overset{\bullet}{U}}\ (Lem.\ 3(i)) \qquad Pre\, U\, t}{t \star (g\, t)' \in U}$$

The second rule:

$$\dfrac{\begin{array}{c}\delta \\ \vdots \\ t \star (g\, t)' \in \overset{\Box}{U}\end{array} \quad (g\, t)' \neq \perp'}{t \star (g\, t)' \in U}\ (Lem.\ 3(iv))$$

Where δ is:

$$\dfrac{\dfrac{g \Subset_{rp} U}{g \subseteq \overset{\Box}{U}} \quad \dfrac{\dfrac{g \Subset_{rp} U}{g \in T_\perp^{in} \to T_\perp^{out'}} \quad t \in T_\perp^{in}}{t \star (g\, t)' \in g}}{t \star (g\, t)' \in \overset{\Box}{U}}$$

The third rule is immediate. □

Theorem 10. $g \Subset_{rp} U \Rightarrow g \Subset_{fp} U$ □

Then, from theorems 9 and 10, we see that the two notions of implementation are equivalent. Hence, so are the two notions of refinement.

7. Conclusions and future work

The model of schemas introduced in $\mathrm{W}_{\bullet}$-refinement not only totalises the schema as a set of bindings, it also introduces the $\perp$ values and extends the domains and co-domains accordingly. The totalisation then stipulates chaotic behaviour outside the precondition and additionally for the $\perp$ values.

Why is it necessary to include the new distinguished values? What are the consequences of totalisation *without* lifting?

Our analysis provides a very clear mathematical explanation for lifting: with non-lifted totalisation it is not possible to prove proposition 5. Note that the proof of that result made explicit use of $\perp$ value. Indeed, we can do better: the following is an explicit counterexample:

Definition 18.

$$\begin{array}{lll} (i) & \overset{\diamond}{U} & =_{df} \{z \in T \mid Pre\ U\ z \Rightarrow z \in U\} \\ (ii) & True & =_{df} [T \mid true] \\ (iii) & Chaos & =_{df} [T \mid false] \end{array}$$

Proposition 20. $\overset{\diamond}{True} = \overset{\diamond}{Chaos}$ □

It is an immediate consequence that the more permissive notion of refinement (based on containment of non-lifted totalised operations according to definition 18(i)) does not, for example, insist that preconditions do not strengthen. We have, however, only begun to provide answers to the natural questions that arise. For example, although lifting appears to be necessary, why does it have to be non-strict with respect to $\perp$? Proposition 20 also raises a question: why is there a distinction between *implicit* (*Chaos*) and *explicit* (*True*) permission to behave? Note that in the Woodcock-completion, $\overset{\bullet}{True} \neq \overset{\bullet}{Chaos}$.

Much the same observation can be made for the other family of refinement theories: again the lifting is critical in preventing the preconditions from strengthening.

Our refinement theories S-refinement and SP-refinement are entirely proof-theoretic, characterising refinement directly in terms of the behaviour of the predicates involved. These are quite closely related to conditions proposed originally by Spivey (these roughly correspond to the premises of our introduction rule for S-refinement). By reformulating this approach as a theory, rather than sufficient conditions, we establish an equivalent framework in which the model extension with the lifting and completions involved are unnecessary. Although we have not shown it here, there are very simple connections between S-refinement and an equivalent theory of refinement based on weakest preconditions: this will be reported in future work.

The approach to specification based on sets of implementations is a well established but somewhat different tradition, and is most usually investigated in a constructive setting. We have demonstrated that what look like radically different models of specification and refinement are, in fact, intimately related.

What we have not reported here is an extension to data refinement in which data simulation relations play a significant role. This is rather in the spirit of [8], except that our investigation is even more general by taking data simulations to be *partial relations*, by default. There is much to say on this topic, but that requires the present work as a necessary precursor. We will, in future work, show that it is possible to formulate S-like theories which are equivalent to generalisations of the W-frameworks (the obvious generalisations are only equivalent for restricted forms of simulation). Moreover, there are interesting results in weakest precondition data refinement to be developed and explored.

Finally, we have not mentioned the implications for the schema calculus. In considering Z as a prime example of a specification language that fits the technical development explored in this paper, one will want to know how the schema operations interact with refinement: in particular, a treatment of monotonicity properties. It is quite well-known that the Z schema calculus has poor monotonicity properties in the relational model. Our results demonstrate that this is not a special feature of the relational model (because all the alternative approaches are equivalent). Indeed, these poor properties are, it seems, intimately linked with the underlying partial relation semantics of Z. One interesting set of approaches which needs to be fully investigated is the consequence of restricting any one of the refinement theories we have outlined here to *atomic* schemas only; and then to redefine the semantics of the schema operators over the new semantics (rather than over partial relations). In this way, refinement would reduce to the subset relation on the semantics and would be fully monotonic. Naturally, the *nature* of the schema algebra would change, but those changes would be very interesting to explore.

8. Acknowledgements

We would like to thank the New Zealand Foundation for Research, Science and Technology (grant reference: UOWX0011) for financially supporting this research. Moshe Deutsch is supported by the British Council through an ORS award. Special thanks for particularly important discussions and comments go to Lindsay Groves, Greg Reeve, David Streader, Ray Turner, Mark Utting and Jim Woodcock.

References

[1] C. Bolton, J. Davies, and J. Woodcock. On the refinement and simulation of data types and processes. In K. Araki, A. Galloway, and K. Taguchi, editors, *Integrated Formal Methods (IFM'99)*. Springer, 1999.

[2] W. P. DeRoever and K. Engelhardt. *Data refinement: model-oriented proof methods and their comparison*. Prentice Hall International, 1998.

[3] J. Derrick and E. Boiten. *Refinement in Z and Object-Z: Foundations and Advanced Applications*. Formal Approaches to Computing and Information Technology – FACIT. Springer, May 2001.

[4] M. Deutsch, M. C. Henson, and S. Reeves. An analysis of total correctness refinement models for partial relation semantics I. *University of Essex, technical report CSM-362 (Submittted to: J. Logic and Computat)*, 2001.

[5] M. Deutsch, M. C. Henson, and S. Reeves. Six theories of operation refinement for partial relation semantics. *University of Essex, technical report CSM-363*, 2002.

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

[7] J. Grundy. *A method of program refinement*. PhD thesis, University of Cambridge, 1993.

[8] J. He, C.A.R Hoare, and J.W. Sanders. Data refinement refined. In G. Goos and J. Hartmanis, editors, *European Symposium on Programming (ESOP '86)*, volume 213 of *Lecture Notes in Computer Science*, pages 187–196. Springer-Verlag, 1986.

[9] M. C. Henson and S. Reeves. Investigating Z. *Journal of Logic and Computation*, 10(1):1–30, 2000.

[10] C.A.R Hoare and J. He. *Unifying Theories of Programming*. Prentice Hall International, 1998.

[11] S. King. Z and the Refinement Calculus. In D. Bjørner, C. A. R. Hoare, and H. Langmaack, editors, *VDM '90 VDM and Z – Formal Methods in Software Development*, volume 428 of *Lecture Notes in Computer Science*, pages 164–188. Springer-Verlag, April 1990.

[12] P. Martin-Löf. Constructive mathematics and computer programming. In *Logic, Methodology and Philosophy of Science VI*, pages 153–175. North Holland, 1982.

[13] S. L. Pfleeger. *Software Engineering - Theory and Practice*. Prentice Hall, 1998.

[14] B. Potter, J. Sinclair, and D. Till. *An introduction to formal specification and Z*. Prentice Hall, 2nd. edition, 1996.

[15] J. M. Spivey. *The Z notation: A reference manual, 2^{nd} ed.* Prentice Hall, 1992.

[16] B. Strulo. How firing conditions help inheritance. In J. P. Bowen and M. G. Hinchey, editors, *ZUM '95: The Z Formal Specification Notation*, volume 967 of *Lecture Notes in Computer Science*, pages 264–275. Springer Verlag, 1995.

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

A. Specification logic - a summary

Our mathematical account takes place in a simple conservative extension $\mathcal{Z}_\mathcal{C}^\perp$ of $\mathcal{Z}_\mathcal{C}$ the core Z-logic of [9]. The only modification we need to make is to include the new distinguished terms which are explicitly needed in the approach taken in [17]. Specifically: the types of $\mathcal{Z}_\mathcal{C}$ are extended to include terms $\perp^T$ for every type T. There are, additionally, a number of axioms which ensure that all the new $\perp^T$ values interact properly, e.g.

$$\frac{}{\perp^{[l_0:T_0 \cdots l_n:T_n]} = \langle\!| \, l_0 \Rrightarrow \perp^{T_0} \cdots l_n \Rrightarrow \perp^{T_n} |\!\rangle}$$

In other words, $\perp^{[l_0:T_0 \cdots l_n:T_n]} \; .l_i = \perp^{T_i} \; (0 \leq i \leq n)$. Note that this is the *only* axiom concerning distinguished bindings, hence, binding construction is *non-strict* with respect to the $\perp^T$ values.

Finally, the extension of $\mathcal{Z}_\mathcal{C}^\perp$ which introduces schemas as sets of bindings and the various operators of the schema calculus is undertaken as usual (see [9]) but the carrier sets of the types must be adjusted to form what we call the *natural carrier sets* which are those sets of elements of types which *explicitly exclude* the $\perp^T$ values:

Definition 19. *The* natural carriers *for each type are defined by closing:*

$$\mathbb{N} =_{df} \{z^{\mathbb{N}} \mid z \neq \perp^{\mathbb{N}} \wedge z = z\}$$

under the operations of cartesian product, powerset and schema set.[2]

As a result the schema calculus is *hereditarily* $\perp$*-free*.

We will also need the *extended carriers*. These are defined for all types as follows:

Definition 20.

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

We indicated in the first section that we can always write the type of operation schemas as $\mathbb{P}(T^{in} \curlyvee T^{out'})$ where T^{in} is the type of the input sub-binding and $T^{out'}$ is the type of the output sub-binding. We also permit *binding concatenation*, written $t_0 \star t_1$ when the alphabets of t_0 and t_1 are disjoint. This is, in fact, exclusively used for partitioning bindings in operation schemas into before and after components, so the terms involved are necessarily disjoint. We lift this operation to sets (of appropriate type):

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

The same restriction obviously applies here: the types of the sets involved must be disjoint.

We will need total functions over types. These are easily introduced.

Definition 21.

$$T_0 \to T_1 =_{df} \{g \in \mathbb{P}(T_0 \star T_1) \mid unicity(g) \wedge total(g)\}$$

Note that functions are modelled as subsets of $T_0 \star T_1$ rather than $T_0 \times T_1$, and that for notational convenience, we let g (*etc.*) range over terms of type $\mathbb{P}(T_0 \star T_1)$. In fact we only do this when such a term is a function.

When g is known to be an element of $T_0 \to T_1$ and $z_0 \in T_0$, we will write $g\ z$ (as usual) for the unique element $z_1 \in T_1$ such that $z_0 \star z_1 \in g$.

[2] The notational ambiguity does not introduce a problem, since only a set can appear in a term or proposition, and only a type can appear as a superscript.