Under consideration for publication in Formal Aspects of Computing

Modular reasoning in Z: scrutinising monotonicity and refinement

Moshe Deutsch¹, Martin C. Henson¹ and Steve Reeves²

¹Department of Computer Science, University of Essex, Wivenhoe Park, Colchester, Essex CO4 3SQ, UK
²Department of Computer Science, University of Waikato, Private Bag 3105, Hamilton, New Zealand
E-mail: {mdeuts, hensm}@essex.ac.uk, stever@cs.waikato.ac.nz

Abstract. The schema calculus operators of Z provide an excellent means for expressing modular specifications but not for undertaking modular reasoning: it is well-known that these operators have poor monotonicity properties. The paper addresses three topics in this context: first, we provide a thorough mathematical analysis of monotonicity with respect to four schema operations and for three notions of operation refinement. Second, we provide a comprehensive analysis of the relational completion operator, known as lifted-totalisation, that underlies the standard notion of refinement in Z. Third, we provide a new semantics which induces a fully monotonic schema calculus.

Keywords: Monotonicity; Operation Refinement; Schema Calculus; Specification Language; Specification Logic

“We used to think that if we knew one, we knew two, because one and one are two. We are finding that we must learn a great deal more about 'and'.” Sir Arthur Eddington (1882-1944)

1. Introduction

This paper addresses three topics:

1. It provides a systematic analysis of the monotonicity properties of four major operators taken from the schema calculus of Z, with respect to the standard notion of (operation) refinement. Failure of monotonicity is well-known, though there is precious little in the way of a formal analysis in the existing literature ([29, 18, 19] are recent exceptions). We isolate conditions under which monotonicity holds;

2. It provides a detailed analysis of the lifted-totalisation relational completion operation for partial relations. The standard model for Z is a partial relation semantics; the standard account of refinement is a total correctness interpretation. The lifted-totalisation completion mediates between the underlying semantics and the interpretation of refinement;

3. It establishes an alternative relational semantics for Z which leads to a fully monotonic schema calculus with respect to refinement.

In the remainder of this introduction we discuss the background and context of our investigations and outline the structure and organisation of the technical sections which follow.

Correspondence and offprint requests to: Martin C. Henson, Department of Computer Science, University of Essex, Wivenhoe Park, Colchester, Essex CO4 3SQ, UK.
1.1. Modular development in Z?

Z [56] is a specification language which permits the modular construction of specifications by means of an algebra of connectives and quantifiers reminiscent of predicate logic. This provides great structural expressivity and is a major reason for the popularity of the approach. Practical examples are covered in the better textbooks (e.g. [4], [65]) and its logical properties, justifying the usual terminology: schema calculus, in [36].

In addition to its use purely as a language for specification, there has been considerable interest in using Z for design and development. An approach which integrates work in refinement with Z is given in, for example, [13, 65, 51, 24]. Yet, a number of issues make this more general use for Z problematic, the most central being the fact that the modular techniques for expressing structured specifications is not accompanied by the possibility of modular reasoning because the schema operations are not monotonic with respect to the standard account of refinement. Thus, refinement takes place on operations expressed as a single schema: the schema calculus operators are removed (essentially by using an equational logic) before applying refinement (there are many examples and case studies, e.g. [61, 43, 42]).

As a consequence, many authors have developed case studies in Z without employing refinement at all (e.g. [49, 53, 31]) or hoping that their work will underlie software development using verification techniques (e.g. [38]). Other authors have addressed this by proposing methods for transforming Z specifications into fully monotonic frameworks such as Morgan’s Refinement Calculus [48, 3] (e.g. [42, 63, 9, 11, 26]) or other notations based on Dijkstra’s guarded command language [23] (e.g. [67, 43]). Unfortunately, these require either the elimination of the schema operators prior to the process or the utilisation of very strong sideconditions. Some solved this difficulty by introducing informal techniques for transforming Z specifications into declarative programming languages such as Haskell (e.g. [39, 27]). Others advocate substituting Z with a more powerful version of Refinement Calculus, which enriches the language with Z-like specification constructors in order to equip it with modularity capabilities (e.g. [58, 44, 28]); ironically, the major disadvantage of this method is that, in many cases, the additional specification constructors re-introduce non-monotonicity. A radically different approach is taken in [33] and [37], where the semantics of both operation schemas (but not state schemas) and the schema calculus is modified in order to attain a language that is both modular and fully monotonic with respect to refinement. This yields a Z-like system in which, however, the schema operators no longer express exactly their usual informal semantics. Unlike the other methods discussed above, this work takes place entirely within the specification language and its logic.

1.2. The partial relation semantics and equational logic of Z

In Z, schemas generally denote partial relations. For example, consider the schema:

$$\text{Predecessor} \equiv \left\{ \left( x, x' : \mathbb{N} \mid x' = x - 1 \right) \right\}$$

The set this denotes contains bindings (valid assignments of values to the observations x and x') of the form:

$$\{ x \equiv n, x' \equiv n - 1 \}$$

for all \( n > 0 \). But no binding of the form:

$$\{ x \equiv 0, x' \equiv m \}$$

for any \( m \in \mathbb{N} \). In this sense the schema is partial when viewed as a relation\(^2\) between its before-state (the sub-binding involving the observation x) and after-state (the sub-binding involving the observation x').

The underlying relation for schemas in Z is not refinement but equality. The definition of the schema operators leads directly to an equational logic. One such equation, characterising conjunction, is:

$$[ D_0 \mid P_0 ] \land [ D_1 \mid P_1 ] = [ D_0 ; D_1 \mid P_0 \land P_1 ]$$

There are similar equations for all schema operators. By orienting these equalities, it is easy to see that every schema expression is trivially equal to a single atomic schema (its normal form, so to speak).

What behaviour is permitted for a correct implementation of a (partial) specification outside the domain of the relation it denotes? To answer this question we need a theory of refinement: a means for comparing such an implementation

---

1 These are based on a method, developed by J. B. Wordsworth, used by IBM UK laboratories at Hursley.

2 We will use U (etc.) in future to range over schema expressions. Strictly speaking these are interpreted as partial relations in the underlying theory ZC; see [36]) or Z\(^C\); (see [21] and a summary in appendix A). Only from section 5.1 will it be necessary to distinguish between them, so until then, for simplicity of presentation, we will write U both for a schema expression (syntax) and for the partial relation (semantics) it denotes.
with such a specification. The standard notion of refinement in Z is based on a subsequent total-relation semantics, known as the lifted-totalised interpretation [65, 13]. Under this interpretation the answer to the question is: anything can happen (this is sometimes known as chaotic behaviour). This is not the only theory of refinement which can be developed on the basis of the underlying partial relation semantics. However, as we shall see in detail, all the various options lead to refinement theories for which, to varying degrees, monotonicity of the schema operators fails.

1.3. Organisation and overview

The paper begins, in section 2, with an overview of operation refinement. We then move on, in section 3, to an analysis of the monotonicity properties of four standard schema calculus operators, each of which is investigated with respect to three notions of operation refinement in Z. Unlike [33] and [37], we pursue our investigation in Z as it is informally understood in, for example, [54] and [65]. A related analysis for schema conjunction and schema disjunction was presented in [29]. We explore those operators in detail and, additionally, schema existential hiding and schema composition. The investigations suggest a number of sideconditions that can be used as “healthiness conditions” for Z specifications, which guarantee monotonicity. The usefulness of these sideconditions is also discussed.

In section 4, the second part of the paper, we investigate the distributivity properties of the lifted-totalisation operator with respect to the four schema operations. We show that none of these operators fully distributes over the lifted-totalisation, uncover reasons for this failure and introduce sideconditions for full-distribution in each case. These investigations motivate the final part of the paper (in section 5): the introduction of a fully monotonic schema calculus based on the lifted-totalised interpretation as the underlying semantics for atomic schemas. The semantics for the schema operations is then given recursively, using the standard operations, and refinement merely becomes the subset relation on the semantics.

Our investigations become possible in virtue of the logic for Z reported in, for example, [36] and benefit from the technique of rendering all the theories of refinement in a proof-theoretic form: as sets of introduction and elimination rules. This leads to a uniform and simple method for proving the various technical results.

We provide some essential notational conventions, and a complete formal account of the theory of preconditions for compound operations, in appendix A;3 this is an important precursor for the investigations presented in sections 3 and 4. The paper concludes with a summary and an agenda for further investigation (section 6).

2. Operation refinement

Operation refinement concerns the derivation of a more concrete operation from a given abstract one, without changing the specification of the underlying state. It is sometimes called algorithm design [67]. 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: what does it mean for one partial relation to refine another? The standard answer involves some sort of lifted-totalisation of the underlying partial relations (see e.g. [65] and [13]). In [21] it is shown that the relational completion notion is equivalent to various other approaches, one of which, S-refinement (and its relatives), is much simpler and more intuitive. Therefore, for the analysis of monotonicity (section 3) we shall work entirely within the simpler S-refinement framework. This permits us to deal directly with partial, rather than with lifted-totalised, relations and, at least at this stage, to avoid the complications involving additional semantic elements.

We begin by introducing three distinct notions of operation refinement in Z, based on three distinct answers to the questions above.

2.1. S-refinement

We introduce a pure proof-theoretic characterisation of refinement, which is closely connected to sufficient refinement conditions introduced by Spivey (hence “S”-refinement) in [54] and as discussed in, for example, [42, 51, 67, 65].

This notion is based on two basic observations regarding the properties one expects in a refinement: first, that a refinement may involve the reduction of nondeterminism; second, that it may also involve the expansion of the domain of definition. Put another way, we have a refinement providing that postconditions do not weaken and that preconditions do not strengthen.

3 This is sufficient for our needs in this paper. Further detail can be found in [36] and [21].
This notion can be captured by forcing the refinement relation to hold exactly when these conditions apply. S-refinement is written \( U_0 \models s U_1 \) (\( U_0 \) S-refines \( U_1 \)) and is given by the definition that leads directly to the following rules:

**Proposition 2.1.** Let \( z_0, z_1 \) be fresh variables.

\[
\begin{align*}
\text{Pre } U_1 \ z \vdash \text{Pre } U_0 \ z \\
\text{Pre } U_1 \ z_0, z_0 \star z'_1 \in U_0 \vdash z_0 \star z'_1 \in U_1 \\
\hline
U_0 \models s U_1 \\
\end{align*}
\]

(\( \sim s^+ \))

\[
\begin{align*}
\text{Pre } U_1 \ t \vdash \text{Pre } U_0 \ t \\
\text{Pre } U_1 \ t_0 \star t'_1 \in U_0 \\
\hline
U_0 \models s U_1 \\
\end{align*}
\]

(\( \sim s^- \))

We prove in [21] and [20] that S-refinement is equivalent to several other characterisations of refinement, such as W₄-refinement based on Woodcock’s chaotic relational completion model [65, 13] (see appendix A, definition A.10). As we remarked above, S-refinement deals directly with the partial relation semantics rather than by means of an interpretation as a (lifted-totalised) relation. It is, therefore, simpler both as a theory and in terms of the analysis we undertake in section 3.

### 2.2. SP-refinement

This is an alternative proof-theoretic characterisation of refinement, which is closely connected to refinement in the behavioural [13] or firing condition [55] approach. This special case of S-refinement may involve reduction of non-determinism but insists on the stability of the precondition. SP-refinement is written \( U_0 \models sp U_1 \) and is given by the definition that leads directly to the following rules:

**Proposition 2.2.** Let \( z, z_0, z_1 \) be fresh variables.

\[
\begin{align*}
\text{Pre } U_1 \ z \vdash \text{Pre } U_0 \ z \\
\text{Pre } U_1 \ z_0, z_0 \star z'_1 \in U_0 \vdash z_0 \star z'_1 \in U_1 \\
\hline
U_0 \models sp U_1 \\
\end{align*}
\]

(\( \sim sp^+ \))

\[
\begin{align*}
\text{Pre } U_1 \ t \vdash \text{Pre } U_0 \ t \\
\text{Pre } U_1 \ t_0 \star t'_1 \in U_0 \\
\hline
U_0 \models sp U_1 \\
\end{align*}
\]

(\( \sim sp^- \))

We showed in [20] that SP-refinement is equivalent to several other characterisations of refinement. For example, W₄-refinement, which is based on the abortive relational completion model (see appendix A, definition A.12) and discussed in [6] and [13].

### 2.3. SC-refinement

SC-refinement is our third alternative proof-theoretic characterisation of refinement. It is written \( U_0 \models sc U_1 \) and is given by the definition that leads directly to the following rules:

**Proposition 2.3.** Let \( z_0, z_1 \) be fresh variables.

\[
\begin{align*}
\text{Pre } U_1 \ z_0, z_0 \star z'_1 \in U_0 \vdash z_0 \star z'_1 \in U_1 \\
\hline
U_0 \models sc U_1 \\
\end{align*}
\]

(\( \sim sc^+ \))

\[
\begin{align*}
\text{Pre } U_1 \ t_0 \star t'_1 \in U_0 \\
\hline
U_0 \models sc U_1 \\
\end{align*}
\]

(\( \sim sc^- \))

**Lemma 2.1.** The following extra rule is derivable for SC-refinement:

\[
\begin{align*}
U_0 \models sc U_1 \\
\text{Pre } U_1 \ t \\
\hline
\text{Pre } U_0 \ t \\
\end{align*}
\]
SC-refinement is introduced for technical reasons which inform the analysis to follow. This notion, in which the precondition may weaken, but in which the postcondition is stable, is not otherwise of much pragmatic interest.

3. Operation refinement and monotonicity in the schema calculus

The major advantage of Z, in contrast to other paradigms such as the Refinement Calculus and even B [2], is its potential for expressing modular specifications using schema operators. However, in order to properly exploit modularity and in particular to undertake specification refinement, it is vital that the various schema operators of the language are monotonic. When monotonicity holds, the components of a composite specification can be refined independently of the remainder of the specification [29]; refinement can then be performed in a modular manner. Unfortunately it is well-known folk-lore that the Z schema calculus operators have very poor monotonicity properties; this has a major effect on their usefulness, for example, in the context of program development from Z specifications.

In this section we analyse the monotonicity properties of four of the most interesting schema calculus operators (conjunction, disjunction, existential hiding and composition) with respect to each one of the refinement theories presented in section 2. We provide examples of monotonicity (or non-monotonicity) and establish sideconditions, as "healthiness conditions" on specifications, in order to attain monotonicity. We also discuss the usefulness of these sideconditions in the context of the various refinement theories we consider.

3.1. Refinement for conjunction

We do not have an introduction rule for the precondition of conjoined operations (see appendix A, section A.3.1). Consequently, schema conjunction is not monotonic with respect to S-refinement. Here is a simple counterexample. Consider the following schemas:

\[ U_0 \equiv [x, x' : \mathbb{N} \mid x' = 8] \qquad U_1 \equiv [x, x' : \mathbb{N} \mid x' < 10] \qquad U_2 \equiv [x, x' : \mathbb{N} \mid x' = 2] \]

We note that \(U_1\) is a nondeterministic operation that can be refined by strengthening its postcondition, for example to \(U_0\). However, when conjoining the operations, we have the following schemas:

\[ U_0 \land U_2 = [x, x' : \mathbb{N} \mid false] \]
\[ U_1 \land U_2 = [x, x' : \mathbb{N} \mid x' = 2] \]

In [21] and [20], we define the chaotic specification as: \(\text{Chaos} \equiv [T \mid false]\). A chaotic specification cannot constitute a refinement of any other specification because this would signify augmentation of undefinedness and would therefore violate any notion of refinement presented in section 2.\(^4\) Thus, \(U_0 \land U_2 \not\subseteq U_1 \land U_2\).

The counterexample also shows the reason for the failure: strengthening the postcondition might create a chaotic specification, due to both the "postcondition only" (single predicate) approach Z takes [47, 60, 59], and the definition of schema conjunction. This motivates the following sidecondition. It is perhaps not surprising that it is precisely the missing introduction rule for the precondition of conjoined operations.

**Proposition 3.1.** Let \(z\) be a fresh variable and \(U_0, U_1, U_2\) be operation schemas with the property that:

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

Then the following rule is derivable:

\[
\frac{U_0 \supseteq U_1 \quad U_2 \supseteq U_1 \land U_2}{U_0 \land U_2 \supseteq U_1 \land U_2}
\]

\(^4\) See [21, section 4.4] for further details.
Proof

\[
\begin{array}{c@{\quad}c@{\quad}c@{\quad}c}
U_0 \sqsupset_\text{a} U_1 & \frac{\text{Pre}(U_0 \land U_2) \in z}{\text{Pre} U_0 z} & \frac{\text{Pre}(U_1 \land U_2) \in z}{\text{Pre} U_1 z} & \frac{\text{Pre}(U_0 \land U_2) \in z}{\text{Pre} U_2 z} \\
\cdots & \cdots & \cdots & \\
U_0 \land U_2 \sqsupset_\text{a} U_1 \land U_2 & \frac{z_0 \star z'_1 \in U_1 \land U_2}{\delta}
\end{array}
\]

Where \( \delta \) is:

\[
\begin{array}{c@{\quad}c@{\quad}c@{\quad}c}
U_0 \sqsupset_\text{a} U_1 & \frac{\text{Pre}(U_1 \land U_2) \in z_0}{\text{Pre} U_1 z_0} & \frac{z_0 \star z'_1 \in U_0}{\text{Pre} U_0 z_0} & \frac{z_0 \star z'_1 \in U_2}{\text{Pre} U_2 z_0} \\
\cdots & \cdots & \cdots & \\
U_0 \land U_2 \sqsupset_\text{a} U_1 \land U_2 & \frac{z_0 \star z'_1 \in U_1 \land U_2}{z_0 \star z'_1 \in U_1 \land U_2}
\end{array}
\]

\( \square \)

Much the same observation can be made for SP-refinement: since non-monotonicity follows by a permissible reduction of nondeterminism, SP-refinement and S-refinement coincide. Proposition 3.1 with \( \sqsupset_\text{a} \) substituted by \( \sqsupset_{\text{sp}} \) holds for SP-refinement; the proof is similar.

SC-refinement guarantees that no reduction of nondeterminism is possible, thus ensuring that schema conjunction is monotonic with respect to SC-refinement.

**Proposition 3.2.** The following rule is derivable:

\[
\begin{array}{c}
U_0 \sqsupset_{\text{sc}} U_1 \\
U_0 \land U_2 \sqsupset_{\text{sc}} U_1 \land U_2
\end{array}
\]

Proof

\[
\begin{array}{c@{\quad}c@{\quad}c@{\quad}c}
\frac{z_0 \star z'_1 \in U_1 \land U_2}{U_0 \sqsupset_{\text{sc}} U_1} & \frac{z_0 \star z'_1 \in U_1 \land U_2}{\frac{z_0 \star z'_1 \in U_0}{U_0 \land U_2 \sqsupset_{\text{sc}} U_1 \land U_2}} & \frac{\delta}{z_0 \star z'_1 \in U_1 \land U_2}
\end{array}
\]

Where \( \delta \) is the \( \delta \) branch of the proof of proposition 3.1 (with \( \sqsupset_\text{a} \) substituted by \( \sqsupset_{\text{sc}} \)). \( \square \)

Note that, although the non-monotonicity of conjunction with respect to S-refinement and SP-refinement is a direct consequence of the ability to strengthen the postcondition, the sidecondition used in proposition 3.1 is applied in the proof branch concerning the precondition. This is not surprising because, if one attempts to prove the refinement of the conjoined schemas given in the counterexample, one discovers that the branch for the postcondition is provable, due to \( \text{false} \) in the antecedent of the implication; whereas the branch for the precondition fails for the opposite reason \( \text{false} \) in the consequent). Thus, one can expect an application of the sidecondition in this branch at some point.

We would like to highlight the value of insights gained from both counterexamples and the formal proofs. One of the benefits of a precise investigation is the ability to deduce or motivate various results as a direct consequence of these. The sidecondition above can be calculated through the direct attempt to prove proposition 3.1.
The solid lines represent the partial relation denoted by the schema $U_1 \lor U_2$, and the dotted lines represent the additional behaviours in the schema $U_0 \lor U_2$. Note the point (marked with a right arrow) which represents *weakening of the postcondition* with respect to $U_1 \lor U_2$. Hence $U_0 \lor U_2 \not\supseteq U_1 \lor U_2$.

### 3.2. Refinement for disjunction

Schema disjunction is *not monotonic* with respect to S-refinement. In contrast to the analysis of schema conjunction in section 3.1, the reason for non-monotonicity in this case is the fact that S-refinement enables us to weaken the precondition. Weakening the precondition of (at least) one component specification might extend the domain of the disjunction of the two components, leading to an *increase in nondeterminism* and thus a failure of refinement.

For example, consider the following schemas:

$$U_0 \equiv [x, x': \mathbb{N} | x' = 2] \quad U_1 \equiv [x, x': \mathbb{N} | x = 0 \land x' = 2]$$

$$U_2 \equiv [x, x': \mathbb{N} | x = 1 \land x' = 3]$$

The specification $U_1$ is partial and, therefore, can be refined to $U_0$ by weakening its precondition. However, respectively disjoining the schemas above yields the following specifications:

$$U_0 \lor U_2 = [x, x': \mathbb{N} | x' = 2 \lor x = 1 \land x' = 3]$$

$$U_1 \lor U_2 = [x, x': \mathbb{N} | x = 0 \land x' = 2 \lor x = 1 \land x' = 3]$$

Clearly $U_0 \lor U_2 \not\supseteq U_1 \lor U_2$. This is because the schema $U_0 \lor U_2$ permits the behaviour $\langle x \mapsto 1, x' \mapsto 2 \rangle$, which is prohibited by $U_1 \lor U_2$. This is a representative example; the only reason why S-refinement can fail in such a case is as a result of an augmentation of nondeterminism with respect to the abstract disjunction; this is shown in Fig. 1. The analysis suggests that the sidecondition will be required in the proof branch concerning the postcondition; as we will now see, this is the case.

**Proposition 3.3.** Let $z$ be fresh and $U_0, U_1, U_2$ be operation schemas with the property that:

$$\text{Pre } U_0 z \land \text{Pre } U_2 z \Rightarrow \text{Pre } U_1 z$$

Then the following rule is derivable:

$$\frac{U_0 \not\supseteq U_1}{U_0 \lor U_2 \not\supseteq U_1 \lor U_2}$$

**Proof**


$$\frac{\text{Pre } (U_1 \lor U_2) z}{(1)} \quad U_0 \not\supseteq U_1 \quad \text{Pre } U_1 z \quad (2) \quad \frac{\text{Pre } U_0 z}{\text{Pre } (U_0 \lor U_2) z} \quad \text{Pre } U_0 z \quad (2) \quad \frac{\text{Pre } U_2 z}{\text{Pre } (U_0 \lor U_2) z} \quad \text{Pre } U_2 z \quad (2) \quad \delta_0 \quad \vdots \quad \frac{z_0 \mapsto z_1 \in U_1 \lor U_2}{U_0 \lor U_2 \not\supseteq U_1 \lor U_2} \quad (1)$$
Where $\delta_0$ stands for the following branch:

\[
\frac{z_0 \star z'_1 \in U_0 \lor U_2 \quad (I)}{z_0 \star z'_1 \in U_1 \lor U_2 \quad \delta_1 \quad (3)} \quad \frac{z_0 \star z'_1 \in T^* \quad (I)}{z_0 \star z'_1 \in U_1 \lor U_2 \quad \delta_1 \quad (3)} \\
\frac{z_0 \star z'_1 \in U_1 \lor U_2 \quad (3)}{z_0 \star z'_1 \in U_1 \lor U_2 \quad (3)}
\]

and $\delta_1$ is:

\[
\frac{z_0 \star z'_1 \in U_0 \quad (3)}{\text{Pre } U_0 \#_0 \quad (4)} \quad \frac{\text{Pre } U_0 \#_2 \quad (4)}{\text{Pre } U_0 \#_0 \land \text{Pre } U_2 \#_0} \\
\frac{\text{Pre } (U_0 \lor U_2) \#_0 \quad (I)}{\text{Pre } U_1 \#_0 \quad (4)} \quad \frac{\text{Pre } U_1 \#_2 \quad (4)}{\text{Pre } U_1 \#_0} \\
\frac{z_0 \star z'_1 \in U_1 \quad (3)}{z_0 \star z'_1 \in U_1 \quad (3)}
\]

\[\square\]

Once again, the sidecondition is determined by the proof: it is precisely the entailment of an otherwise unprovable proposition $\text{Pre } U_1 \#_0$ from the available assumptions $\text{Pre } U_0 \#_0$ and $\text{Pre } U_2 \#_0$.

The situation with SC-refinement is very similar since, as the counterexample illustrates, the critical factor leading to non-monotonicity in this case is weakening of the precondition. SC-refinement sanctions this and, therefore, proposition 3.3 holds with $\exists_0$ substituted by $\exists_{sc}$. The proof, given this substitution, is similar.

Schema disjunction is monotonic with respect to SP-refinement because weakening of the precondition is prohibited. The following rule is derivable:

**Proposition 3.4.**

\[
U_0 \exists_{sp} U_1 \\
U_0 \lor U_2 \exists_{sp} U_1 \lor U_2
\]

**Proof**

\[
\frac{\delta_0 \quad \delta_1 \quad \text{Pre } (U_0 \lor U_2) \#_0 \quad (I)}{z_0 \star z'_1 \in U_1 \lor U_2 \quad (1)}
\]

Where $\delta_0$ is identical to the precondition branch in the proof of proposition 3.3 (with $\exists_0$ substituted by $\exists_{sp}$) and $\delta_1$ stands for the following branch:

\[
\frac{z_0 \star z'_1 \in U_0 \lor U_2 \quad (I)}{z_0 \star z'_1 \in U_1 \lor U_2 \quad (3)} \quad \frac{z_0 \star z'_1 \in T^* \quad (I)}{z_0 \star z'_1 \in U_1 \lor U_2 \quad (3)} \quad \frac{z_0 \star z'_1 \in U_0 \lor U_2 \quad (I)}{z_0 \star z'_1 \in U_1 \lor U_2 \quad (3)}
\]

\[\square\]

### 3.3. Refinement for existential quantification

There is, of course, an intimate relationship between disjunction and existential quantification. We might expect the monotonic properties of schema existential quantification to be similar to those for schema disjunction. Indeed, schema existential quantification is *not monotonic* with respect to S-refinement because weakening of the precondition might admit behaviours in the concrete operation that are unacceptable in the abstract. The reason is that schema existential quantification can hide an arbitrary observation and, in particular, observations that can lead to an augmentation of nondeterminism. This is shown by the counterexample in Fig. 2.
We present the specifications $U_0$ and $U_1$, whose alphabets comprise the boolean observations $x$, $x'$, $y$ and $y'$. The specifications $\exists x, x' : \mathbb{B} \cdot U_0$ and $\exists x, x' : \mathbb{B} \cdot U_1$ hide the pair of observations $x$ and $x'$ from $U_0$ and $U_1$. Note that the schema $U_1$ denotes a partial operation and $U_0$ S-refines it by weakening the precondition. Nevertheless, hiding those observations introduces a weakening of the postcondition (marked with a right arrow in Fig. 2) of $\exists x, x' : \mathbb{B} \cdot U_0$ with respect to $\exists x, x' : \mathbb{B} \cdot U_1$. Hence, S-refinement fails.

Like schema disjunction, the above counterexample also suggests that, in order to prove monotonicity of existential quantification is not monotonic with respect to S-refinement.

**Proposition 3.5.** Let $x$ be fresh and $U_0$, $U_1$ be operation schemas with the property that:

$$\text{Pre } U_0 x \Rightarrow \text{Pre } U_1 x$$

Then the following rule is derivable:

$$\exists z : T^z \cdot U_0 \trianglerighteq \exists z : T^z \cdot U_1$$

**Proof**

$$U_0 \trianglerighteq U_1 \quad \frac{\text{Pre } U_0 y}{\text{Pre } U_1 y} \quad \frac{\text{Pre } (\exists z : T^z \cdot U_0) y}{\text{Pre } (\exists z : T^z \cdot U_1) y} \quad y = z \quad \frac{\text{Pre } (\exists z : T^z \cdot U_0) z}{\text{Pre } (\exists z : T^z \cdot U_1) z} \quad \frac{\text{Pre } (\exists z : T^z \cdot U_0)}{\text{Pre } (\exists z : T^z \cdot U_1)} \quad \frac{\exists z : T^z \cdot U_0 \trianglerighteq \exists z : T^z \cdot U_1}{\exists z : T^z \cdot U_0 \trianglerighteq \exists z : T^z \cdot U_1} \quad (1)$$

Where $\delta_0$ is:

$$\exists z : T^z \cdot U_0 \trianglerighteq \exists z : T^z \cdot U_1$$

$$\frac{\text{Pre } U_0 w}{\text{Pre } U_1 w} \quad \frac{\text{Pre } (\exists z : T^z \cdot U_0) w}{\text{Pre } (\exists z : T^z \cdot U_1) w} \quad w = z_0 \quad \frac{\text{Pre } (\exists z : T^z \cdot U_0) z_0}{\text{Pre } (\exists z : T^z \cdot U_1) z_0} \quad \frac{\exists z : T^z \cdot U_0 \trianglerighteq \exists z : T^z \cdot U_1}{\exists z : T^z \cdot U_0 \trianglerighteq \exists z : T^z \cdot U_1} \quad (4)$$

$$\exists z : T^z \cdot U_0 \trianglerighteq \exists z : T^z \cdot U_1$$

$$\frac{\text{Pre } U_0 z}{\text{Pre } U_1 z} \quad \frac{\text{Pre } (\exists z : T^z \cdot U_0) z}{\text{Pre } (\exists z : T^z \cdot U_1) z} \quad z_0 = z_{\delta_0} \quad \frac{\exists z : T^z \cdot U_0 \trianglerighteq \exists z : T^z \cdot U_1}{\exists z : T^z \cdot U_0 \trianglerighteq \exists z : T^z \cdot U_1} \quad (3)$$
and $\delta_1$ is:

$$\delta_1 = \begin{align*}
\mathbf{Pre} U_0 \mathbf{y}_0 \\
\quad \vdots \\
\mathbf{Pre} U_1 \mathbf{y}_1 \\
\mathbf{y}_1 \in U_1 \\
\mathbf{y}_1 \in U_0
\end{align*}
$$

Note that the above sidecondition forces a “fixed preconditi on” refinement, which is precisely SP-refinement. So it is
an immediate consequence that schema existential hiding is monotonic with respect to SP-refinement:

**Proposition 3.6.** The following rule is derivable:

$$U_0 \sqsubseteq_{sp} U_1 \Rightarrow \exists z : T^2 \bullet U_0 \sqsubseteq_{sp} \exists z : T^2 \bullet U_1$$

The proof is essentially identical to proposition 3.5.

One might expect that the monotonicity properties of existential hiding with respect to S-refinement and SC-refinement
would coincide. Indeed, the sidecondition of proposition 3.5 is required for proving monotonicity of schema existential
hiding with respect to SC-refinement. However, since the sidecondition guarantees stability of the precondition, and
SC-refinement the stability of the postcondition, the result holds only when the abstract and concrete operations are
equivalent, which is of course, far from useful.

### 3.4. Refinement for composition

It is not surprising that schema composition is *not monotonic* with respect to S-refinement, because composition in $Z$
can be expressed in terms of *conjunction* and *existential quantification* (appendix A, definition A.7(iii)). Consideration
of the results of sections 3.1 and 3.3 suggests that both weakening the precondition and strengthening the postcondition
of the component specifications will cause a problem. This is fairly intuitive since reduction of nondeterminism is
demonic with respect to schema composition (since strengthening the postcondition on the left of the composition
might demonically choose those after-states that do not connect with the precondition of the operation to the right of
the composition). This results in losing requirements from the domain of the composed specifications. In addition,
weakening the precondition on the right of the composition might demonically extend the specification domain in
such a way that the composition will introduce unacceptable behaviours mapped from the original precondition.

The following counterexample illustrates the problems and motivates a solution by suggesting a sidecondition. Con-
sider the following specifications:
We introduce two abstract specifications, \( U_1 \) and \( U_3 \), and their respective S-refinements \( U_0 \) and \( U_2 \). We then show that the composition of the underlying concrete specifications does not constitute a refinement of the composition of their abstract counterparts. We label the before-states of each specification using the labels \( x, y, z \) and \( w \) (from the top).

The specification \( U_1 \) has two instances of nondeterminism and \( U_0 \) S-refines it by reducing both of these. However, this is demonic, so that the after-state mapped from \( w \) in \( U_0 \) does not compose with anything in the precondition of \( U_2 \); we therefore lose the before-state \( w \) from the domain of \( U_0 \upharpoonright U_2 \), whereas it still exists in the domain of \( U_1 \upharpoonright U_5 \). This is one of the reasons for non-refinement.\(^5\) \( U_2 \) S-refines \( U_3 \) by weakening its precondition, so in conjunction with the demonic reduction of nondeterminism by \( U_0 \), the after-state mapped from \( x \) in \( U_0 \) is composed with the before-state \( y \) in \( U_5 \). This introduces a binding mapping from \( x \) in \( U_1 \upharpoonright U_2 \) which does not exist in \( U_1 \upharpoonright U_3 \), yet \( x \) is in the precondition of \( U_1 \upharpoonright U_3 \). This is the second reason for non-refinement. The fact that \((
abla S)\) fails for reasons concerning both the precondition and postcondition suggests that neither the SP nor SC refinement theories will support this monotonicity result. Furthermore, it suggests that a sidecondition which is sufficient for proving monotonicity with respect to S-refinement will be needed in both the precondition and postcondition branches of the proof; as we shall see, this is indeed the case.

The counterexample above, and \([30, p.39-40]\),\(^6\) suggest a remedy: if we insist that every after-state in the range of \( U_1 \) is mapped onto at least one value in the precondition of \( U_3 \), then, not only can strengthening the postcondition (on the left) by \( U_0 \) never be demonic (as, in the presence of refinement, the precondition of \( U_2 \) is at least as large as the one of \( U_3 \)), but also weakening the precondition (on the right) by \( U_2 \) can never introduce an after-state in \( U_0 \upharpoonright U_2 \) that is connected via an intermediate value that was not in the precondition of \( U_2 \).\(^7\) The property is strong connectivity and it is defined as follows:

**Definition 3.1 (Strong connectivity).**

\[
\text{Sc } U_0 \ U_1 = \forall z_0, z_1 \bullet z_0 \uparrow z_1' \in U_0 \Rightarrow \text{Pre } U_1 \ z_1
\]

We can prove that schema composition is monotonic with respect to all three refinement theories, providing \( \text{Sc } U_1 \ U_3 \) holds; but we can do better than that. Although strong connectivity is a very intuitive sidecondition, there is a weaker condition which is also sufficient. This can be motivated by considering further counterexamples. We call it forking connectivity. Two specifications comply with this property if, for every nondeterministic before-state (forking point) in the first specification, either all the after-states mapped from it connect with some before-state in the precondition of the second specification, or none of them does.

\(^5\) Note that had the precondition of \( U_1 \) not been weakened by \( U_2 \), we would have also lost the before-state \( x \) from the precondition of \( U_0 \upharpoonright U_2 \). This would have induced a chaotic specification \( U_0 \upharpoonright U_2 \).

\(^6\) Grundy proposes a modified definition of composition, in which strong connectivity is embedded.

\(^7\) Unless, of course, this is as a result of composing an after-state in \( U_0 \) that constitutes a new behaviour (as a consequence of weakening the precondition of \( U_1 \)) with a before-state outside the precondition of \( U_3 \) but which is inside the precondition of \( U_2 \). Such a case is not relevant in the present context.
Definition 3.2 (Forking connectivity).

\[ \text{Fc } U_0 U_1 \equiv_{df} \forall z_0, z_1, z_2 \; (z_0 \star z'_1 \in U_0 \land z_0 \star z'_2 \in U_0 \land \text{Pre } U_1 z_1) \Rightarrow \text{Pre } U_1 z_2 \]

Obvious introduction and elimination rules follow from this.

With this in place, we can now prove the monotonicity result. We shall provide only the proof for S-refinement.

Proposition 3.7. Let \( U_0, U_1, U_2 \) and \( U_3 \) be operation schemas with the property that:

\[ \text{Fc } U_1 U_3 \]

Then the following rule is derivable:

\[
\begin{array}{c}
U_0 \supseteq_s U_1 \quad U_2 \supseteq_s U_3 \\
U_0 \supseteq_s U_1 \quad U_2 \supseteq_s U_3
\end{array}
\]

Proof

\[
\begin{array}{c}
\text{Pre } \left( U_1 \supseteq_s U_3 \right) z \\
\text{Pre } \left( \text{Fc } U_1 U_3 \right) z
\end{array}
\]

Where \( \alpha_0 \) stands for the following branch:

\[
\begin{array}{c}
U_0 \supseteq_s U_1 \\
\text{Pre } \left( U_1 \supseteq_s U_3 \right) z
\end{array}
\]

and \( \delta_1 \) is:

\[
\begin{array}{c}
U_0 \supseteq_s U_1 \\
\text{Pre } U_1 z
\end{array}
\]

Where \( \alpha_1 \) stands for the following branch:

\[
\begin{array}{c}
\text{Pre } \left( U_1 \supseteq_s U_3 \right) z_0 \\
\text{Pre } \left( \text{Fc } U_1 U_3 \right) z
\end{array}
\]

Finally, it is interesting to note that, since weakening the precondition causes a problem on the right and strengthening
the postcondition causes a problem on the left, it is an immediate consequence that schema composition is monotonic on the right with respect to SP-refinement (because the precondition is fixed), and is monotonic on the left with respect to SC-refinement (because the postcondition is fixed). Hence, the following rules are derivable:

**Proposition 3.8.**

\[
\frac{U_0 \sqsupseteq \text{sp} \ U_1}{U_2 \sqsupset U_0 \sqsupseteq \text{sp} \ U_2 \sqsupset U_1} \quad \frac{U_0 \sqsupseteq \text{sc} \ U_1}{U_0 \sqsupset U_2 \sqsupseteq \text{sc} \ U_1 \sqsupset U_2}
\]

**Proof.** We provide only the proof for SP-refinement.

\[
\begin{align*}
\text{Pre} (U_2 \sqsupset U_1) z & \quad (1) \\
\text{Pre} (U_2 \sqsupset U_0) z & \quad (2) \\
\text{Pre} (U_0 \sqsupset U_1) y & \quad \delta \\
\text{Pre} (U_2 \sqsupset U_0) z & \quad (2) \\
\text{Pre} (U_0 \sqsupset U_1) y & \quad \delta \\
z_0 \star z'_1 \in U_2 \sqsupset U_1 & \quad (1)
\end{align*}
\]

Where \( \delta \) is:

\[
\begin{align*}
z_0 \star z'_1 \in U_2 \sqsupset U_0 & \quad (3) \\
z_0 \star z'_1 \in U_1 & \quad (3)
\end{align*}
\]

\( \square \)

4. **Distributivity properties of the chaotic relational completion operator**

The standard interpretation of refinement for Z in the literature (e.g. [65, 13]) is what we have called \( W_{\star} \)-refinement [21, 20]:

\[
U_0 \equiv_{w_{\star}} U_1 \overset{\text{df}}{=} U_0 \subseteq U_1
\]

where \( U \) is the lifted-totalisation of \( U \) (see appendix A, section A.4 for further detail).

It is important to note that this definition concerns the partial relation interpretation of schema expressions. That is, the interpretation of schemas, and of all the operations for building modular specifications, are logically prior to the theory of refinement.

\( W_{\star} \)-refinement is, as we have already remarked, equivalent to S-refinement: the theory we used in the previous section. So everything we have established so far also applies to \( W_{\star} \)-refinement. It is, however, often illuminating to consider matters through distinct though equivalent formulations; this section is devoted to that, mainly through a consideration of lifted-totalisation as an operator in its own right.

One way of illustrating the failure of monotonicity, as it arises in the \( W_{\star} \)-refinement framework, is to take a look at how the lifted-totalisation interacts directly with the schema operators. For example, if the following full distributivity property held:

\[
(U_0 \star U_1) = U_0 \land U_1
\]

then schema conjunction would be fully monotonic with respect to refinement. That is, we would have:

\[
\frac{U_0 \equiv_{w_{\star}} U_2 \quad U_1 \equiv_{w_{\star}} U_3}{U_0 \land U_1 \equiv_{w_{\star}} U_2 \land U_3}
\]
with proof:

\[
\frac{z \in (U_0 \land U_1)}{z \in U_0 \land U_1} \quad (I)
\]

Here, the proof annotations indicate the problem: only half of the full distributivity equation holds. Put another way, for full (unconditioned) monotonicity, we needed the equation at the level of the total-relation semantics, but in Z we have it only at the level of the partial relation semantics (this is the usual equational logic to be found in the textbooks). The situation is similar for every schema operator. In this section we will analyse in detail the reasons why full distributivity of the lifted-totalisation operator fails with respect to each of the schema calculus operators investigated in section 3. We will introduce sideconditions that are sufficient to attain full distributivity equations and then analyse their usefulness and their relationship to the sideconditions introduced in section 3.

4.1. Distributivity for conjunction

The problem with distributing the lifted-totalisation operator over schema conjunction arises when identical before-states of the two component specifications do not agree on their after-states. This leads only to a distributivity inequality and not to full equivalence. The case is illustrated in Fig. 4. The figure illustrates two specifications, \( U_0 \) and \( U_1 \), which share the before-state \( w \), but map it to distinct after-states. Conjoining these two specifications removes \( w \) from the domain and the completion operator interprets partiality as chaos: anything is possible (marked with a right arrow in Fig. 4). This contrasts with the result of conjoining the completions of the specifications, which introduces the partiality at the level of the refinement theory. Here the partiality looks more like infeasibility. General infeasibility is often known (e.g. in two-predicate frameworks such as the Refinement Calculus and VDM [40, 41]) as magic:\(^8\) an

\(^8\) For a complete account of extreme specifications see, for example, [48], [30], [60] and [64].
extreme specification whose precondition is true and whose postcondition is false (it is guaranteed to terminate, yet must establish an impossible outcome). In the figure, the infeasibility is localised: we will refer to this as local magical behaviour.  

As we can see in Fig. 4, distributing the relational completion operator over schema conjunction may cause local magical behaviour whenever distinct after-states are mapped from the same before-state (ω in this case) in the two component specifications, prior to the lifted-totalisation. This behaviour results in partiality, in a similar fashion to the two-predicate based frameworks (including the approach taken in [33] and [37]). Note that Z, a single-predicate framework, is capable only of modelling two of the extreme specifications: chance and chaos, in which the preconditions and postconditions are simultaneously, true and false, respectively. For this reason, we have only a distributivity inequation:

**Proposition 4.1.** The following rule is derivable:

\[
\begin{align*}
  t_0 \star t_1' &\in \hat{U}_0 \land \hat{U}_1 \\
  t_0 \star t_1' &\in (U_0 \land U_1)
\end{align*}
\]

□

The only way to ensure that distributivity holds in the other direction is by preventing such contentious states in the component specifications: preventing local magical behaviour. This is achieved by insisting that the conjunction of the two specifications will, at least, retain the precondition of their disjunction:

**Definition 4.1 (Properly conjoined operation schemas).**

\[
P_c U_0 U_1 \equiv \forall z \cdot \text{Pre} (U_0 \lor U_1) z \Rightarrow \text{Pre} (U_0 \land U_1) z
\]

**Proposition 4.2.** Let \( U_0 \) and \( U_1 \) be operation schemas with the property that:

\[
P_c U_0 U_1
\]

Then the following rule is derivable:

\[
\begin{align*}
  t_0 \star t_1' &\in (U_0 \land U_1) \\
  t_0 \star t_1' &\in \hat{U}_0 \land \hat{U}_1
\end{align*}
\]

□

### 4.2. Distributivity for disjunction

The lifted-totalisation operator does not fully distribute over schema disjunction because completing component specifications, which have different preconditions, may induce chaotic behaviour in their disjunction, but non-chaotic behaviour when the component specifications are disjoined and then completed. Fig. 5 illustrates this.

---

9 Since the appearance of partiality arises here because of over-constraining the compound specification, rather than because of an under-constraint (e.g. the partiality in Predecessor), there is a good argument for treating it differently (see sections 5.5 and 5.6).
The specification $U_0$ has just one after-state mapped from $x$ and the specification $U_1$ has just one after-state mapped from $y$. Disjoining these partial relations, prior to completion, results in those two after-states (mapped from $x$ and $y$ in $U_0 \lor U_1$) and chaotic behaviour everywhere else. However, applying the lifted-totalisation to $U_0$ and $U_1$ (individually), gives rise to chaotic behaviour mapped from $y$ in $U_0$ and similarly for $x$ in $U_1$. Thus, $U_0 \lor U_1$ is chaotic from these two before-states (marked with right arrows); hence, we have an inequation rather than a full equivalence.

**Proposition 4.3.** The following rule is derivable:

$$t_0 \star t'_1 \in (U_0 \circ U_1)$$

$$t_0 \star t'_1 \in U_0 \lor U_1$$

□

We observed, in Fig. 5, that full distributivity fails, because of distinctions in the preconditions of the specifications $U_0$ and $U_1$. Therefore, insisting that the component specifications have identical preconditions guarantees full distributivity:

**Definition 4.2 (Stable preconditions).**

$$Sp U_0 U_1 = df \forall z \bullet Pre U_0 z \Leftrightarrow Pre U_1 z$$

**Proposition 4.4.** Let $U_0$ and $U_1$ be operation schemas with the property that:

$$Sp U_0 U_1$$

Then the following rule is derivable:

$$t_0 \star t'_1 \in (U_0 \circ U_1)$$

$$t_0 \star t'_1 \in U_0 \lor U_1$$

□
4.3. Distributivity for existential quantification

Fully distributing the relational completion operator over schema existential hiding fails. This is because hiding observations after applying lifted-totalisation can introduce chaotic behaviour that will not always arise when hiding observations before lifted-totalisation. This is shown in Fig. 6.

We present a specification $U$ whose alphabet comprises the boolean observations $x$, $x'$, $y$ and $y'$. Hiding the observations $x$ and $x'$ yields a total specification $\exists x, x' : B \cdot U$. The only effect of lifted-totalisation on this will be the mapping of $\bot$ onto all after-states. Yet, hiding the same observations after lifted-totalisation introduces chaotic behaviour from the before-state $T$ (marked with a right arrow) in the specification $\exists x, x' : B \cdot \tilde{U}$. This is a consequence of mapping the before-state $FT$ (which is outside the precondition of $U$) onto all the after-states in $\tilde{U}$ (including $\bot$). As a result, hiding the observations $x$ and $x'$ in $\tilde{U}$ leaves the remainder of this state sanctioning every possible outcome. For this reason, we have only a distributivity inequation:

**Proposition 4.5.** The following rule is derivable:

\[
\frac{t \in (\exists z : T^\sharp \cdot U)}{t \in \exists z : T^\sharp \cdot \tilde{U}}
\]

\[\square\]

Since existential quantification is a generalisation of disjunction, it is not surprising that the failure of the converse inequation is reminiscent of the case for schema disjunction (section 4.2), although here we have one specification rather than two: the difference described by Fig. 6 arises because distinct before-states, which involve the same hidden observation in the specification, have different precondition status.\(^{10}\)

Naturally, the remedy is very similar to “stable preconditions” (definition 4.2), though here we have only a single specification: we need to ensure that any before-state in the precondition of $\exists z : T^\sharp \cdot U$ is equivalently in the precondition of $\tilde{U}$. One direction is merely $(\text{Pre}_x^\exists)$ (see appendix A, section A.3.3); thus all we need is the following property:

**Definition 4.3 (Weak binding).** $Wb \ U =_{df} \forall x \cdot \text{Pre} (\exists z : T^\sharp \cdot U) x \Rightarrow \text{Pre} \ U x$

\(^{10}\) One is in the precondition and one is outside the precondition of the specification.
Proposition 4.6. Let $U$ be operation schema with the property that:

$Wb U$

Then the following rule is derivable:

\[
\frac{t \in \exists z : T^z \cdot U}{t \in (\exists z : T^z \cdot U)}
\]

\[\square\]

4.4. Distributivity for composition

The lifted-totalisation operator distributes over schema composition (but not conversely) because composing a non-deterministic specification (on the left) with a partial specification (on the right) may give rise to local chaos in the composition of their (individual) completions, but which might not arise in the completion of their composition. This is illustrated in Fig. 7.

![Fig. 7. Lifted-totalisation does not fully distribute over schema composition.](image)

An observation, such as $w$, outside the precondition of $U_1$ plays no part in linking before-states of $U_0$ and after-states of $U_1$ in the composition $U_0 \circ U_1$ (nor its lifted-totalisation). Thus $x$, in $U_0 \circ U_1$, is associated with the two after-states, mapped from $z$ in $U_1$. However, applying the relational completion operator to $U_0$ and $U_1$ separately, results in chaotic behaviour mapped from $w$ in $\U_1$ and consequently chaotic behaviour (marked with a right arrow) from $x$ in $\U_0 \circ \U_1$. Therefore, we have only the following inequation. We provide the proofs in this case.

Proposition 4.7. The following rule is derivable:

\[
t_0 \star t'_1 \in (U_0 \circ U_1)
\]

Proof

\[
\frac{t_0 \star y' \in U_0}{(L.A.3(i))}
\]

\[
\frac{y \star t'_1 \in U_1}{(L.A.3(i))}
\]

\[
\frac{t_0 \star t'_1 \in (U_0 \circ U_1)}{(3)}
\]

\[
\frac{t_0 \star t'_1 \in U_0 \circ U_1}{(I)}
\]

\[
\frac{t_0 \star t'_1 \in \U_0 \circ \U_1}{(3)}
\]

\[
\frac{t_0 \star t'_1 \in \U_0 \circ \U_1}{(I)}
\]
Where \( \delta_0 \) stands for the following branch:

\[
\begin{align*}
\neg \text{Pre } (U_0 \uplus U_1) \quad (1) \\
\neg \text{Pre } U_0 \land (\forall z \cdot t_0 \star \star z' \in U_0 \Rightarrow \neg \text{Pre } U_1 z) \\
t_0 \star t'_1 \in U_0 \uplus U_1 \\
\end{align*}
\]

Where \( \delta_1 \) is:

\[
\begin{align*}
t_0 \star t'_1 \in (U_0 \uplus U_1) \\
\neg \text{Pre } U_0 \land \beta_0 \\
l_0 \star w' \in U_0 \\
w \star t'_1 \in U_1 \\
\end{align*}
\]

and \( \delta_2 \) is:

\[
\begin{align*}
t_0 \star t'_1 \in (U_0 \uplus U_1) \\
\neg \text{Pre } U_0 \land \beta_0 \\
l_0 \star w' \in U_0 \\
w \star t'_1 \in U_1 \\
\end{align*}
\]

Where \( \beta_0 \) is:

\[
\begin{align*}
\forall z \cdot t_0 \star \star z' \in U_0 \Rightarrow \neg \text{Pre } U_1 z \\
l_0 \star w' \in U_0 \Rightarrow \neg \text{Pre } U_1 w \\
\neg \text{Pre } U_1 w \\
\end{align*}
\]

and \( \beta_1 \) is identical to \( \delta_1 \) modulo the substitution of the label \( \beta_1 \) for the label \( \beta_0 \).

There are two observations we can make of the proof. First, note that the proof step labelled \( \star \) denotes an application of \( (\star) \) but where the definition A.10 is expressed using disjunction (in the obvious way) in place of implication, leading to a single disjunctive elimination rule. This rule is also used in the proofs for propositions 4.3 and 4.5. Second, the proof depends on use of the law of excluded middle. We suspect that this result is strictly classical, and there appear to be many other similar examples in refinement theory (e.g. [15, 14, 16]), so abandoning the constructive approach for \( Z \) taken in, for example, [34, 35] and [46] may be inevitable.

Fig. 7 demonstrates that distributivity fails in the other direction precisely because the two after-states mapped from \( x \) in \( U_0 \) coincide with two before-states in \( U_1 \) with different precondition status. We can see that the forking point \( y \) in \( U_0 \) does not constitute a problem, since \( w \) and \( k \) are both outside the precondition of \( U_1 \); thus \( y \) is associated with chaotic behaviour in both \( (U_0 \uplus U_1) \) and \( U_0 \uplus U_1 \). Furthermore, had we had \( w \) in the precondition of \( U_1 \), we would have obtained the same (non-chaotic) behaviour associated with \( x \) in both cases. This suggests that a sidecondition guaranteeing full distributivity would insist on associating all the after-states, mapped from a certain nondeterministic before-state in \( U_0 \), with some before-states in \( U_1 \) – all of which have the same precondition status. This is, indeed, the forking connectivity property used to ensure monotonicity of schema composition with respect to S-refinement (section 3.4).
Proposition 4.8. Let $U_0$ and $U_1$ be operation schemas, with the property that:

$$Fc \ U_0 \ U_1$$

Then the following rule is derivable:

$$t_0 \star t'_1 \in (U_0 \circledast U_1)$$

Proof

$$\frac{t_0 \star t'_1 \in U_0 \circledast U_1}{t_0 \star t'_1 \in (U_0 \circledast U_1)}$$

Where $\delta$ stands for the following branch:

$$\frac{t_0 \star t'_1 \in U_0 \circledast U_1}{t_0 \star t'_1 \in (U_0 \circledast U_1)}$$

$$(1)$$

and $\beta$ stands for the following branch:

$$\beta$$

$$\frac{t_0 \star t'_1 \in U_0 \circledast U_1}{t_0 \star t'_1 \in (U_0 \circledast U_1)}$$

$$(2)$$

4.5. Discussion

The sideconditions we have isolated, either for ensuring full distributivity or for establishing monotonicity directly, are not similar to the syntactic sideconditions routinely associated with logical rules, such as $\exists$-elimination. Syntactic sideconditions are decidable, so it can always be determined when a rule applies. The sideconditions we have formulated are proof-theoretically more complex and, in fact, only semi-decidable. From a practical point of view this is not very satisfactory. Moreover, most make mention of the concrete specification in addition to the abstract specification. This is unfortunate because it reduces the practical use of the refinement rules when used calculationally: to construct the concrete specification. The exception to this are the two connectivity principles used in the case of composition. And one could in fact avoid mention of the concrete specification, in the case of disjunction, by omitting one of the antecedent propositions: thereby obtaining a condition which is purely abstract, but which is, of course, somewhat less applicable. It could be argued that these sideconditions are reasonable only if they refer exclusively to either the abstract or the concrete specifications [57]. In this way the true spirit of abstraction (in which the internal structure of the abstract specification is not to be disclosed) is upheld. Overall the lack of monotonicity is a distinct drawback which such proof-theoretic sideconditions do not address satisfactorily from a practical perspective.

Our analysis has of course concentrated on only two (equivalent) notions of refinement: S-refinement and $W\cdot -$refinement. Possibly, there are other formulations of refinement which would be better behaved. And there are, in fact, several alternative approaches:

- **Weakest precondition refinement** – it is possible to reinterpret the partial relations in terms of a weakest precondition semantics and to characterise refinement in the standard way in that regime;
• Sets of implementation – in the spirit of constructive theories of program development, e.g. Martin-Löf Type Theory [45] (though in the setting of classical logic) it is possible to reinterpret specifications as sets of permissible implementations. Refinement in this case is simply set inclusion;

• Strict-lifted-totalisation – it is possible to modify the lifted-totalisation so that the lifting is strict (abortive) rather then non-strict (chaotic);

• Non-lifted-totalisation – it is possible to totalise the partial relations without lifting if one is prepared to exclude fully chaotic behaviour from the notion of precondition.

We demonstrated in [21] that all of these theories of refinement are equivalent to the standard lifted-totalised account. As a consequence, all suffer from the same weaknesses in terms of their (lack of) monotonicity properties. Naturally, one could ask: are there still others which have yet to be discovered? In addressing this question we would need to find some principles which distinguish a relation worthy of the name “refinement” from any arbitrary binary relation on specifications. After all, the schema operators are all fully monotonic with respect to equality, but equality is evidently not a notion of refinement. In capturing the general principles one will be led to the properties described by S-refinement or SP-refinement. A notion of refinement ought to be at least sound with respect to one or other of those. Our analysis already demonstrates the limits of modular reasoning with respect to these notions.

The analysis of distributivity in this section does include an interesting clue which motivates our final section. This was the observation that we have an equational logic at the level of the underlying partial relations, but not at the level of the total relations involved in refinement. This suggests an alternative approach. Instead of developing a schema calculus at the level of partial relations and only then introducing total relational refinement, we could introduce that calculus afterwards. This would naturally lead to the subset relation on the modified relational model. Of course it would lead to a distinct schema calculus with possibly quite different properties. These would also need to be investigated. The latter investigation is beyond the scope of this paper; but the results of the following section suggest that it would be interesting to pursue.

5. A fully monotonic schema calculus

Z, as we have seen, takes a layered approach to refinement: the underlying semantics of schema expressions is partial relational, but refinement is based on a subsequent interpretation. This may take many guises (including the standard, lifted-totalisation model) but all lead to equivalent theories for which the schema operators are not monotonic. Z does however have an equational logic (section 1.2).

This leaves us with an interesting question: Which is more important: the equational logic or monotonicity of the schema operations? – evidently we cannot have both simultaneously. Historically, the clear answer is: the equational logic, though this may be because, until very recently, the lack of a mathematical analysis precluded addressing what was otherwise an unasked question. Obviously, there is no definitive answer: the matter would have to be settled by features of the application context. However, since one answer to the question has been thoroughly investigated (that is Z, as we understand it), it will be interesting to explore the consequences of the alternative answer: if we abandon the equational logic we may be able to rehabilitate monotonicity.

We introduce, in this section, one possible approach for attaining a fully monotonic schema calculus. Motivated by the distributivity results in the preceding section, we replace the usual partial relation semantics with the lifted-totalised interpretation, taking it directly as the semantics for atomic schemas and then introducing the meaning of compound operation schema expressions by recursion over their structure, using the standard interpretation of the schema operators. In this way, refinement reduces to the subset relation on the new semantics, and the schema calculus becomes fully monotonic.

5.1. Logic and semantics

The new semantics (written \[\llbracket \cdot \rrbracket_1\]) is defined as follows:

Definition 5.1. Let \( U \) be an atomic operation schema, then:

\[ \llbracket U \rrbracket_1 =_{df} U \]

Extending this to all schema expressions is compositional: as given by definition A.8 for the standard interpretation.
5.2. \( T_*\)-refinement

Refinement based on the new totalised semantics (hence “\( T_*\)-refinement) is written \( U_0 \sqsupset U_1 \) and is simply defined as a subset relation on the new semantics. The term “totalised” here refers to the approach taken for the interpretation of atomic schemas: under-specification is modelled in this theory of refinement as chaos. Note that schema expressions in general still denote partial relations because we use the standard interpretation of conjunction and composition. Over-specification is modelled in this theory of refinement as magic (see section 4.1 for our earlier discussion of this).

Definition 5.2.

\[
\begin{align*}
U_0 \sqsupset U_1 \iff & U_0 \subseteq U_1 \\
\end{align*}
\]

The following introduction and elimination rules are derivable for \( T_*\)-refinement:

Proposition 5.1. Let \( z \) be a fresh variable.

\[
\begin{align*}
\frac{z \in U_0 \quad z \in U_1}{U_0 \sqsupset U_1} (\exists^*_z) \\
\frac{t \in U_0 \quad t \in U_1}{U_0 \sqsupset U_1} (\forall^*_t)
\end{align*}
\]

\( \square \)

5.2.1. Refinement logic

In this model we lose the usual equational logic for the operation schema calculus, trading this for refinement inequations. Put another way: the various semi-distributivity results presented earlier (section 4) become, under this model, an inequational logic.

Proposition 5.2 (Conjunction inequation).

\[
\begin{align*}
[D_0 \mid P_0] \land [D_1 \mid P_1] \sqsupset [D_0; D_1 \mid P_0 \land P_1] \\
\end{align*}
\]

Proof Follows by proposition 4.1 and [36] – proposition 5.5. \( \square \)

Proposition 5.3 (Disjunction inequation).

\[
\begin{align*}
[D_0; D_1 \mid P_0 \lor P_1] \sqsupset [D_0 \mid P_0] \lor [D_1 \mid P_1] \\
\end{align*}
\]

Proof Follows by proposition 4.3 and [36] – proposition 4.10. \( \square \)

Proposition 5.4 (Existential quantification inequation).

\[
\begin{align*}
[D \mid \exists u : T^x \bullet P[u]] \sqsupset [\exists z : T^z ; D \mid P] \\
\end{align*}
\]

Proof Follows by proposition 4.5 and [36] – proposition 4.11. \( \square \)

Proposition 5.5 (Composition inequation). Let \( T^{in} = [x : T] \) and \( T^{out'} = [x' : T] \).

\[
\begin{align*}
[x, x' : T \mid \exists v \cdot T^v \bullet P_0[x'/v'.x] \land P_1[x/v.x]] \sqsupset [x, x' : T \mid P_0] ; [x, x' : T \mid P_1] \\
\end{align*}
\]

Proof Follows by propositions 4.7 and A.2. \( \square \)

5.3. Monotonicity

All the schema operators defined for the new semantics in section 5.1 are monotonic with respect to \( T_*\)-refinement. The proofs for the following monotonicity properties are trivial: they all follow because the semantics fully distributes over the standard relational operations.

Proposition 5.6. All four schema operators are monotonic with respect to \( T_*\)-refinement.

\[
\begin{align*}
\frac{U_0 \sqsupset U_1}{U_0 \land U_2 \sqsupset U_1 \land U_2} (i) \\
\frac{U_0 \sqsupset U_1}{U_0 \lor U_2 \sqsupset U_1 \lor U_2} (ii)
\end{align*}
\]
5.4. The relationship between $W_•$-refinement and $T_•$-refinement

What is the relationship between $T_•$-refinement and the standard characterisation of refinement in $Z$, (what we have denoted as) $W_•$-refinement?

The answer follows from the investigation conducted in section 4: since $T_•$-refinement relies on distributing the semantics over the standard schema algebra, $W_•$-refinement would be equivalent to $T_•$-refinement only for those schema expressions satisfying the conditions for full distributivity of the lifted-totalisation over the standard operations. We will now formalise this relationship, highlighting the specific use of the sideconditions (section 4) in the process.

In order to manage both interpretations of schemas simultaneously, we need to distinguish between membership in the two schema calculi. To this end we will use $∈_0$ in the standard theory and $∈_1$ in the new semantics. More exactly, we have the standard model (see appendix A):

$⟦t ∈_0 U⟧_0 = \text{df} \: ⟦t⟧_0 ∈ ⟦U⟧_0$

and, for the new interpretation:

$⟦t ∈_1 U⟧_1 = \text{df} \: ⟦t⟧_1 ∈ ⟦U⟧_1$

Proposition 5.7. Let $U$ range over only those schema expressions which satisfy the following:

(i) Schema disjunction expressions satisfy stable preconditions (definition 4.2);

(ii) Existentially quantified schema expressions satisfy weak binding (definition 4.3);

(iii) Schema composition expressions satisfy forking connectivity (definition 3.2).

Then the following rule is derivable:

\[
\frac{t ∈_0 U}{t ∈_1 U}
\]

Proof By induction over the structure of $U$ using propositions 4.1, 4.4, 4.6 and 4.8. □

Proposition 5.8. Let $U$ range over only those schema expressions for which schema conjunction subexpressions are properly conjoined (definition 4.1). Then the following rule is derivable:

\[
\frac{t ∈_0 U}{t ∈_1 U}
\]

Proof By induction over the structure of $U$ using propositions 4.2, 4.3, 4.5 and 4.7. □

Proving that $W_•$-refinement implies $T_•$-refinement (under certain circumstances) is now straightforward:

Theorem 5.1. Let $U_0$ range over that subset of schema expressions satisfying the sideconditions stated in proposition 5.7 and let $U_1$ range over that subset of schema expressions satisfying the sidecondition stated in proposition 5.8. Then the following is derivable:

\[
\frac{U_0 \ni_{i_0} U_1}{U_0 \ni_{i_1} U_1}
\]

□

Likewise:

Theorem 5.2. Let $U_0$ range over a subset of schema expressions, which satisfy the sidecondition stated in proposition 5.8 and let $U_1$ range over a subset of schema expressions satisfying the sideconditions stated in proposition 5.7. Then the following is derivable:

\[
\frac{U_0 \ni_{i_0} U_1}{U_0 \ni_{i_1} U_1}
\]
Together, theorems 5.1 and 5.2 establish that the theories of $T_\ast$-refinement and $W_\ast$-refinement are equivalent for schema expressions satisfying full distributivity of the relational completion operator with respect to the various schema operations.

5.5. Discussion

Apart from the lack of full distributivity with respect to the schema algebra (section 4), the standard interpretation has an additional flaw: recall that, as mentioned in section 4.1, the standard theory of refinement for $Z$ interprets all partiality chaotically, as under-specification.

Consider the following example:

$U_0 \equiv \{ x, x' : \mathbb{N} | x' = 1 \}$  
$U_1 \equiv \{ x, x' : \mathbb{N} | x' = 2 \}$  
$U_2 \equiv \{ x, x' : \mathbb{N} | x' = 8 \}$

The conjunction of $U_0$ and $U_1$, given the equational logic, is equivalent to:

$U_0 \land U_1 = \{ x, x' : \mathbb{N} | x' = 1 \land x' = 2 \}$

This model is nowhere defined and, given that the standard notion of refinement in $Z$ is merely a subset relation on the totalisations of the (atomic) specifications, this is Chaos which may be refined by any specification [30]. Therefore we have:

$U_2 \supseteq_w U_0 \land U_1$

Yet, is this a reasonable refinement?

Imagine a safety-critical specification, for which two independent assessments of certain inputs must combine (conjunctively) to create safe conditions on certain outputs. If circumstances arise in which those independent assessments can conflict, then it will be possible to make arbitrary refinement choices in conditioning the outputs at that point of conflict. When the system is sufficiently complex as to require machine support for its refinement, a mathematically correct refinement (indeed a machine-checked system development) might result in unnoticed absurdities and, of course, the possibility that the final system is anything but safe. It should be stressed that this is a flaw in the conceptualisation and not in the mathematical account (which is of course perfectly sound): mathematical correctness is, in itself, an insufficient guarantee.

Note that, interpreted in the new semantics, the inequation fails; indeed, since refinement is simply subset on the model, partiality can never be removed by refinement. In the new theory of refinement, partiality is treated chaotically in those cases which arise from under-specification (in atomic schemas) and treated magically in cases where it arises from over-specification (in some instances of conjunction or composition). In particular, since it is not possible to implement magic [47, 48, 30], nothing, including $U_2$, can refine $U_0 \land U_1$.

5.6. Some alternative approaches

An alternative approach, suggested by Dunne [25], is based on the abortive-lifted-totalisation, which we denote as $\bigcirc$ (see appendix A, definition A.12 and [20]); this is effectively $T_{\bigcirc}$-refinement, that is, refinement based on the general approach described above, but where definition 5.1 is replaced by:

$$\llbracket U \rrbracket_2 =_{df} \bigcirc U$$

The abortive-lifted-totalisation operator is characterised by the same distributivity properties as its chaotic counterpart (section 4) and hence $T_{\bigcirc}$-refinement provides a similar inequational logic to the one for $T_\ast$-refinement (section 5.2.1) only with an underlying abortive semantics. Of course $T_{\bigcirc}$-refinement precludes the weakening of preconditions, which is, for many applications, a weakness and is not one shared by the model we introduced earlier.

In light of the analysis provided in [21], [15] and [17, 14], in which the strict-lifted-totalisation, denoted $\bigcirc$ (appendix A, definition A.11), has the same effect in model-theoretic refinement as its non-strict counterpart, one may be inclined to suppose that $T_{\bigcirc}$-refinement\(^\text{11}\) is equivalent to $T_\ast$-refinement. In fact the strict-lifted-totalisation does not share

---

\(^{11}\) $T_{\bigcirc}$-refinement is based on the model where definition 5.1 is replaced by: $\llbracket U \rrbracket_1 =_{df} \bigcirc U$. 

distributivity properties with the other two models and, as a result, $T_{\circ}$-refinement and $T_{\bullet}$-refinement are not equivalent. This difference stems from the fact that, unlike $T_{\circ}$ and $T_{\bullet}$, $\bot$ in $T_{\circ}$ maps only to the after-state $\bot$, whereas any other before-state outside the precondition maps chaotically. Note that in the analysis of section 4, the only result requiring an explicit use of lemma A.3(iii), indicating the non-strictness of $\bot$, is proposition 4.7; hence it is bound to fail in the strict model. In fact, none of the other results crucially involve $\bot$, so they all trivially hold in the strict model.

The reason for losing the unconditioned distributivity equation for composition in this model is demonstrated by Fig. 8. We introduce a partial specification $U_0$ (the before-state $y$ is outside its precondition) and a total specification $U_1$. The partiality of $U_0$ induces partiality for their composition, thus $y$ is locally chaotic in $(U_0 \circ U_1)$ (as well as in its non-strict counterpart). Therefore, in order for the distributivity inequation to hold, it is essential that $y$ is locally chaotic in $U_0 \circ U_1$. Attaining this when distributing the non-strict-lifted-totalisation is trivial, because $y$ is mapped onto $\bot$ (as well as to everything else) in $U_0$ and $\bot$ is mapped onto all the after-states in $U_1$. Hence, in any case, $y$ will be locally chaotic in $U_0 \circ U_1$. In contrast, local chaos might not be produced when distributing the strict model over composition (marked with a right arrow in Fig. 8). This occurs because $U_0$ is partial and $U_1$ is total but not surjective. Hence, the remedy is the following sidecondition:

**Definition 5.3 (Totality dependency).**

$$Td U_0 U_1 = def \text{ total}(U_1) \Rightarrow \text{ total}(U_0) \lor \text{ surjective}(U_1)$$

Where total and surjective are defined as usual.

In which case:

$$(U_0 \circ U_1) \subseteq U_0 \circ U_1$$

In fact, the distributivity inequations for composition are counter-dependent in the strict model: when one inequation fails the other necessarily succeeds. This is a direct consequence of the property that: $\text{ total}(U_1) \Rightarrow Fc U_0 U_1$.

However, the above sidecondition is very strong and the lack of an unconditioned inequation for composition in the strict model has an important consequence: we lose the refinement inequation for composition (proposition 5.5) in the $T_{\circ}$-refinement model.\(^{12}\) It would be very interesting to explore the pragmatic consequences of that.

\(^{12}\) In line with the results in [21, 15, 17, 14], this would be the only reason to prefer Woodcock’s non-strict model to its strict counterpart.
6. Conclusions and future work

This paper provides a thorough investigation of monotonicity properties for the schema calculus of Z, with respect to a variety of notions of operation refinement.

In the first and second parts we carefully examined four schema calculus operators, looking at monotonicity properties directly and at distribution properties with respect to the lifted-totalisation operation which lies at the heart of the standard account of refinement in Z. In addition to demonstrating the lack of monotonicity properties, we provided a number of sideconditions which are sufficient for guaranteeing monotonicity – although these conditions are proof-theoretic, rather than syntactic, in character and, therefore, arguably of limited use.

The third part of the paper introduced a novel semantics for schemas which is trivially monotonic. In exchange for monotonicity we must give up the usual equational logic, trading it for an inequational logic of refinement. In many ways this is unexceptional and unobjectionable: most formal methods take refinement as the basic relation; Z is unusual in insisting on equality and, as we have seen, it is this priority for equality which is responsible for the lack of useful modular properties for refinement in the standard account. Possibly the reason for this is essentially historical: Z was one of the first methods to be established and it was some time before refinement entered the scene. Moreover, it appears that it was originally conceived as essentially a simple notation for presenting mathematical descriptions, with a distinct emphasis on syntax rather than logic or semantics. Similarly, the schema operations were essentially pragmatic, rather than theoretic, in character and, therefore, arguably of limited use.

Naturally, the loss of the usual equational logic means that the schema operations, in the new semantics, have entirely different properties from those they have in the standard account. Quite apart from the theoretical questions, some of which we have addressed here, are equally important pragmatic questions. How does one use an alternative schema calculus to construct specifications? What is a useful family of such operations? Does the theory of refinement make practical sense? This last question is one we have considered in considerable detail for the standard account and found it lacking: first, because modular specification is not accompanied by modular reasoning (failure of monotonicity) and second, for the reasons discussed in section 5.5 (the potential folly of interpreting over-specification (magic) as under-specification (chaos)).

The second question, concerning appropriate operations, is also important. After all, there is no reason to suppose that these would be exactly those of Z, and for at least two reasons: first, because the semantics is different, and second, because the very purpose of the schema operators is generalised beyond specification in a framework which stresses refinement: it now includes issues in design and implementation. The space of possibilities here is quite large, and the territory relatively unexplored. There remains, then, a number of theoretical and pragmatic questions which map out interesting avenues for future detailed investigation.

Exploration of specification combinators in other, or mixed, frameworks has an established history. Ward in [58] examines specification constructs in the Refinement Calculus, including conjunction and disjunction, though these are not monotonic. That work extends [42] which is possibly the earliest examination of connections between Z and the Refinement Calculus together with issues of schema algebra. More recently Paige [50] examined Z and other formalisms within the context of heterogeneous frameworks, motivated by method integration. Here, it would be possible to translate Z schemas into, for example, predicative specifications [32] accessing better monotonicity properties as a consequence. But the translations are at the level of atomic schemas, so any Z structure must, in general, be removed. Having said that, Ward [58] does provide a few specialised laws which maintain structure, which would extend to the heterogeneous framework of Paige: these include refining disjunctions directly to alternations and conjunctions directly to compositions, under strong sideconditions.

There are other approaches which consider program development directly from Z specifications, the most ambitious being ZRC [11, 9]. In this approach Z is given a WP-semantics (provided in [10]) equivalent to its (then) standard semantics (see [8]) and this is used to integrate specification with Refinement Calculus. The passage from Z specifications to specification statements induces preconditions in the standard way: as feasibility conditions. The use of schema operators is hampered (there are quite strong sideconditions on rules involving schema operators) because the Refinement Calculus solves the frame-problem by insisting that observations outside this frame do not change (this is a trivial consequence of the WP-semantics) and this does not sit well with component schemas which have overlapping or disjoint frames. Despite these problems the approach is well-developed and has much to recommend it: the approaches described in [37, 18] (which specifically address the disadvantages just listed, but doubtless have limitations of their own) can be carefully compared in the future with ZRC.

13 Consider, for example, the role of sequencing in Refinement Calculus.
7. Acknowledgements

We would like to thank the New Zealand Foundation for Research, Science and Technology (references: UOWX0011 and UOWX0202) and the Royal Society of Great Britain, for financially supporting this research. Moshe Deutsch is supported by the British Council through an ORS award.

This work has been influenced in its development by too many people to name explicitly. However, special thanks for particularly important discussions and comments go to Rob Arthan, Steve Dunne, Lindsay Groves, Greg Reeve, David Streader, Ray Turner and Mark Utting.

References

A. Specification logic - a synopsis

In this appendix, we will revise a little Z logic, settling our notational conventions in the process. The reader may wish to consult [36] and [21] for a more leisurely treatment of our notational and meta-notational conventions.

Our analysis takes place in $\mathbb{Z}_C^+$ [21], which is a simple conservative extension of the “Church-style” version of the Z-logic, $\mathbb{Z}_C$ [36], which incorporates $\bot$ into its types. This provides a satisfactory logical account of the schema calculus of Z as it is normally understood, and permits a formalisation of the various relational completion operators we consider.

A.1. Schemas

$\mathbb{Z}_C^+$ is a typed theory in which the types of higher-order logic are extended with schema types whose values are unordered, label-indexed tuples called bindings. For example, if the $T_i$ are types and the $z_i$ are observations (constants) then:

$$[\ldots z_i : T_i \ldots ]$$

is a (schema) type. Values of this type are bindings, of the form:

$$\{ \ldots z_i \mapsto t_i^{T_i} \ldots \}$$

where $t_i^{T_i}$ means that the term $t_i$ has type $T_i$. Binding selection, written $t.x$, is axiomatised so that, for example:

$$\{ x \mapsto 2, y \mapsto 3 \} . x = 2$$

Selection generalises so that $t.P$ denotes the predicate $P$ in which each observation $x$ is replaced by $t.x$. Filtered bindings play a major role in the schema calculus. Such terms have the form $t \upharpoonright T$ and are axiomatised so that, for example:

$$\{ x \mapsto 2, y \mapsto 3 \} . [x : b] = \{ x \mapsto 2 \}$$

The symbols $\leq, \land, \lor$ and $\rightarrow$ denote the schema subtype relation, and the operations of schema type intersection and (compatible) schema type union and schema type subtraction. We let $U$ (with diacriticals when necessary) range over schema expressions. When $U$ is an operation schema, these are sets of bindings linking, as usual, before observations with after observations. This captures the informal account to be found in the literature (e.g. [24], [65]). We can always write the type of such operation schemas as $\mathbb{F}(T^\text{in} \lor T^\text{out})$ where $T^\text{in}$ is the type of the “before” sub-binding (state) and $T^\text{out}$ is the type of the “after” sub-binding. We also permit binding concatenation, written $b_0 \upharpoonright b_1$, when the alphabets of $b_0$ and $b_1$ are disjoint. We lift this operation to sets (of appropriate type), with obvious introduction and elimination rules, by means of:

**Definition A.1.**

$$C_0 \star C_1 = df \{ z_0 \star z_1 | z_0 \in C_0 \land z_1 \in C_1 \}$$

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

We introduce two notational conventions in order to avoid the repeated use of filtering in the context of membership and equality propositions.

**Definition A.2.** Let $T_1 \leq T_0$.

$$t_0^{T_0} \in C^\upharpoonright T_1 = df \ t \upharpoonright T_1 \in C$$

**Definition A.3.** Let $T_1 \leq T_0$ or $T_0 \leq T_1$.

$$\frac{t_0^{T_0} = t_1^{T_1} = df \ b_0 \upharpoonright (T_0 \land T_1) = t_1 \upharpoonright (T_0 \land T_1)}{t_0^{T_0} = t_1^{T_1}}$$

$\mathbb{Z}_C^+$ includes distinguished terms which are needed in the approach taken in [65] for completing relations. Specifically: the types include terms $\bot^T$ for every type $T$. There are, additionally, a number of axioms which ensure that all the new $\bot^T$ values interact properly, for example:

**Axiom A.1.**

$$\bot^T \upharpoonright [a_0 \upharpoonright a_0] = df \{ z_0 \mapsto \bot^T \ldots z_n \mapsto \bot^T \}$$

$\square$
In other words, $\bot^{[z_0 \ldots z_n]}_{\bot^{T_i}} \ni z_i = \bot^{T_i} (0 \leq i \leq n)$. Note that this is the only axiom concerning distinguished bindings, hence, binding construction is non-strict with respect to the $\bot^{T}$ values. We show in [21] that $\mathcal{Z}_{C}^{\perp}$ is a conservative extension of the original Z logic $\mathcal{Z}_{C}$.

A.2. The schema calculus in $\mathcal{Z}_{C}^{\perp}$

The definitional extension $\mathcal{Z}_{C}^{\perp}$ of $\mathcal{Z}_{C}$, which introduces schemas as sets of bindings and the various operators of the schema calculus, is undertaken as usual (see [36]) but, in $\mathcal{Z}_{C}^{\perp}$, 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 that explicitly exclude the $\bot^{T}$ values:

Definition A.4. Natural carriers for each type are defined by closing: $\mathbb{N} \models_{df} \{ z \in \mathbb{N} \mid z \neq \bot^{\mathbb{N}} \}$ under the operations of cartesian product, powerset and schema set.\(^{14}\)

Definition A.5 (Semantics for atomic schemas). $\llbracket T \mid P \rrbracket_0 =_{df} \{ z \in T \mid z \neq P \}$

Note that this definition\(^{15}\) draws bindings from the natural carrier of the type $T$. As a consequence, writing $t(\bot)$ for a binding satisfying $t \cdot \bot = \bot$ for some observation $\bot$, we have:

**Lemma A.1.**

\[
\begin{align*}
\frac{t(\bot) \in U}{\text{false}}
\end{align*}
\]

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

Definition A.6.

\[
T_\perp =_{df} T \cup \{ \bot \}
\]

In a similar manner to [36], we now define four operations in $\mathcal{Z}_{C}^{\perp}$, which pave the way for the interpretation of the schema algebra in $\mathcal{Z}_{C}^{\perp}$. They are polymorphic: the type of the expression is determined by the particular sets to which they are applied. Note that we use the meta-variable $C$ to range over sets (terms of type $\mathbb{P}$ $T$). $T$ denotes the set $T^{in} \star T^{out'}$ and $T^*$ denotes the set $T^{in} \star T^{out}$.

Definition A.7. Let $z$ be a fresh variable in all the four definitions below.

**Conjunction and disjunction.** Let $T^{in}$ be $T^{in}_0 \vee T^{in}_1$ and $T^{out'}$ be $T^{out}_0 \vee T^{out}_1$.

\[
\begin{align*}
(i) & \quad C^{P}_{0} \wedge C^{P}_{1} =_{df} \{ z \in T^* \mid z \in C_0 \wedge z \in C_1 \} \\
(ii) & \quad C^{P}_{0} \vee C^{P}_{1} =_{df} \{ z \in T^* \mid z \in C_0 \vee z \in C_1 \}
\end{align*}
\]

**Composition.**

We will deal with instances of composition where the expression $C_0 \ni C_1$ has the type $\mathbb{P}(T^{in}_0 \vee T^{in}_1)$ and where $C_0$ is of type $\mathbb{P}(T^{in}_1 \vee T^{out}_1)$ and $T^{out}_0 = T^{in}_1$. With all this in place we can omit the type superscripts when analysing composition in the paper.

\[
\begin{align*}
(iii) & \quad C_{0}^{P(T^{in}_0 \vee T^{in}_1)} \wedge C^{P}_{1}^{P(T^{in}_1 \vee T^{out}_1)} =_{df} \{ z_0 \star z'_1 \in T^{in}_0 \star T^{out}_1 \mid \exists y \in T^{out} \ni x_0 \star y \in C_0 \wedge y \star z'_1 \in C_1 \}
\end{align*}
\]

**Existential quantification.** Let $T_x =_{df} \{ z : T^x \}$. If $C$ has the type $\mathbb{P}$ $T$ and $T_x \leq T$, then an expression of the form $\exists z : T^x \bullet C$ will always have the type $\mathbb{P}(T - T_x)$; thus, we can omit the type superscripts when analysing existential hiding in the paper.

\[
\begin{align*}
(iv) & \quad \exists z : T^x \bullet C^{P \perp} =_{df} \{ x \in (T - T_x)^* \mid \exists y \in T \bullet y \in C \wedge x = y \uparrow (T - T_x) \}
\end{align*}
\]

With all these in place, we can now interpret schema expressions in $\mathcal{Z}_{C}^{\perp}$:

\[^{14}\] 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.

\[^{15}\] We write this interpretation as $\llbracket T \rrbracket_0$ so as to distinguish it from the ones used in section 5.1 in the main body of the paper.
We then immediately get the following derived rules for the various schema operators in $Z^*_C$:

**Proposition A.1. Schema conjunction.** Let $i \in 2$.

$$
\frac{t \in U_0 \quad t \in U_1}{t \in U_0 \land U_1} \quad (U^*_0) \quad \frac{t \in U_0 \land U_1}{t \in U_1} \quad (U^*_1) \quad \frac{t \in U_0 \land U_1}{t \in T^*} \quad (U^*_2)
$$

**Schema disjunction.** Let $i \in 2$.

$$
\frac{t \in T^* \quad t \in U_0 \lor U_1}{t \in U_0 \lor U_1} \quad (U^*_0) \quad \frac{t \in U_0 \lor U_1}{t \in U_0 + P} \quad (U^*_0) \quad \frac{t \in U_0 \lor U_1}{t \in U_1 + P} \quad (U^*_1) \quad \frac{t \in U_0 \lor U_1}{t \in T^*} \quad (U^*_2)
$$

**Schema composition.**

$$
\frac{t_0 \star y' \in U_0 \quad y \star t'_1 \in U_1}{t_0 \star t'_1 \in U_0 + U_1} \quad (U^*_0) \quad \frac{t_0 \star t'_1 \in U_0 + U_1}{t_0 \star y' \in U_0, y \star t'_1 \in U_1 + P} \quad (U^*_0) \quad \frac{t_0 \star t'_1 \in U_0 + U_1}{t_0 \star t'_1 \in T_0 + T_1} \quad (U^*_2)
$$

**Schema existential hiding.**

$$
\frac{t \in U}{t \in \exists z : T^* \bullet U} \quad (U^*_1) \quad \frac{t \in \exists z : T^* \bullet U}{t \in \exists z : T \bullet P} \quad y \in U, y \equiv t + P \quad (U^*_2) \quad \frac{t \in \exists z : T^* \bullet U}{t \in (T - T_1)^*} \quad (U^*_3)
$$

The usual sideconditions apply to the eigenvariable $y$ in both $(U^*_0)$ and $(U^*_1)$. □

**Proposition A.2 (Composition logic).** Notice that the above rules for schema composition constitute a major simplification to the ones provided in [36]. We require a simplification of the equational logic for schema composition.

Let $T^{in} = [x : T]$ and $T^{out} = [x : T_1]$ then the following equation is derivable for schema composition:

$$
[x, x': T | P_0]_T \equiv [x, x': T | P_1] = [x, x': T | P_0[x'/v.x] \land P_1[x/v.x]]
$$

□

The schema calculus in $Z^*_C$ preserves the meaning of the schema calculus in $Z_C$. This is established by induction over the structure of schema expressions, which shows that the $Z^*_C$ natural carriers and the $Z_C$ carrier sets lead to equivalent schema calculi. Hence, the $Z^*_C$ schema calculus is hereditarily $\perp$-free.

### A.3. Preconditions

We can formalise the idea of the **precondition** of an operation schema, that is, the domain of the relation between the before-states and after-states that the schema denotes.

**Definition A.9.** Let $T^{in} \leq V$.

$$
Pre \ U x^V = af \ \exists z \in U \bullet x \equiv z
$$

**Proposition A.3.** Let $y$ be a fresh variable, then the following introduction and elimination rules are immediately derivable for preconditions:

$$
\frac{t_0 \in U \quad t_0 \equiv t_1}{Pre \ U \ t_1} \quad \frac{Pre \ U \ t \quad y \in U, y \equiv t + P}{P}
$$

□
In general the precondition of an operation schema will not be the whole of \( T^m \). In this sense operation schemas denote partial relations.

A key to reasoning about the monotonicity properties of the various schema calculus operators is the necessity of reasoning about the precondition of operations defined by schema operations. This topic has been investigated informally in, for example, [62] and [67] (for conjoined and disjoined operation schemas); and formally, for the schema calculus generally, in [19].

We now provide complete precondition theories for schema conjunction, schema disjunction, schema existential hiding and schema composition. We shall not provide the proofs for the various results; should the reader be interested, a complete account for these is provided in [22].

A.3.1. The precondition for conjunction

In general, the precondition of a conjunction of operations is not the conjunction of the preconditions of the individual components [62]. This is a direct consequence of the underlying “postcondition only” approach Z takes.

In fact, we can be more precise: the usual form of a conjunction introduction rule fails, whereas the elimination rules hold. An example, which embodies this result, can be found in [62]. This can be remedied using a strong syntactic sidecondition, for example insisting that the alphabets of the operations are disjoint (see, for example, [29] and [67, p.214]).

**Proposition A.4.** Let \( i \in 2 \), then the following elimination rules are derivable for the precondition of conjoined schemas:

\[
\frac{\text{Pre} \left( U_0 \land U_1 \right) t}{\text{Pre} U_i t} \quad (\text{Pre}_i^\land)
\]

\[
\frac{\text{Pre} U_0 t \lor \text{Pre} U_1 t}{\text{Pre} \left( U_0 \lor U_1 \right) t} \quad (\text{Pre}_i^\lor)
\]

**A.3.2. The precondition for disjunction**

The precondition of schema disjunction is better behaved. This is due to the fact that existential quantification is disjunctive; that is, fully distributes over disjunction (see [62], [65, p.210] and [67, p.125]). Hence, we get the following results:

**Proposition A.5.** Let \( i \in 2 \), then the following introduction and elimination rules for the precondition of the disjunction of schemas are derivable:

\[
\frac{\text{Pre} U_i t}{\text{Pre} \left( U_0 \lor U_1 \right) t} \quad (\text{Pre}_i^\lor)
\]

\[
\frac{\text{Pre} \left( U_0 \lor U_1 \right) t \quad \text{Pre} U_0 t \lor \text{Pre} U_1 t}{P} \quad (\text{Pre}_i^\lor)
\]

With these in place, we can easily prove the full distributivity of the precondition over disjunction (this is also stated in [64]).

**Theorem A.1.**

\[
\text{Pre} \left( U_0 \lor U_1 \right) t \Leftrightarrow \text{Pre} U_0 t \lor \text{Pre} U_1 t
\]

**A.3.3. The precondition for existential quantification**

The following rules are derivable:

**Proposition A.6.**

\[
\frac{\text{Pre} U t}{\text{Pre} \left( \exists z : T^a \cdot U \right) t} \quad (\text{Pre}_i^\exists)
\]

\[
\frac{\text{Pre} \left( \exists z : T^a \cdot U \right) t \quad \text{Pre} U y, y = t t \lor \text{Pre} U y}{P} \quad (\text{Pre}_i^\exists)
\]

Note that the usual sideconditions apply to the eigenvariable \( y \). ⊐

**A.3.4. The precondition for composition**

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

**Proposition A.7.**

\[
\frac{t_0 \cdot t_1' \in U_0 \quad \text{Pre} U_1 t_1}{\text{Pre} \left( U_0 \circ_1 U_1 \right) t_0 \quad (\text{Pre}_i^\circ)}
\]

\[
\frac{\text{Pre} \left( U_0 \circ_1 U_1 \right) t_0 \quad \text{Pre} U_1 y, t_0 \circ y' \in U_0 \lor P}{P} \quad (\text{Pre}_i^\circ)
\]

The usual sideconditions apply to the eigenvariable \( y \). ⊐
Lemma A.2. The following additional rule is derivable for the precondition of composition:

\[
\frac{\text{Pre} (U_0 \downarrow U_1) t_0}{\text{Pre} U_0 t_0}
\]

□

A.4. Relational completion

In this section, we review three relational completion operators discussed in [21] and [20]. We begin by defining the non-strict-lifted-totalisation in line with the intentions described in [65], chapter 16.

Definition A.10 (Non-strict-lifted-totalisation).

Let

\[
U = \{ z_0 \star z_1 \in T^\star \mid \text{Pre} U z_0 \Rightarrow z_0 \star z_1 \in U \}
\]

Then the following introduction and elimination rules are derivable:

Proposition A.8.

\[
\begin{align*}
\frac{t_0 \star t_1' \in T^\star \quad \text{Pre} U t_0 \quad t_0 \star t_1' \in U}{t_0 \star t_1' \in \dot{U}} \quad \text{(i)} \quad \frac{t_0 \star t_1' \in \dot{U} \quad \text{Pre} U t_0}{t_0 \star t_1' \in U} \quad \text{(ii)} \\
\frac{t' \in T'^{\text{out}}}{t' \in \dot{U}} \quad \text{(iii)} \quad \frac{\neg \text{Pre} U t \quad t \in T'^{\text{in}} \quad t' \in \dot{U}}{t \star t' \in \dot{U}} \quad \text{(iv)} \quad \frac{\neg \text{Pre} U t_0 \quad t_0 \in T'^{\text{in}} \quad t_1' \in T'^{\text{out}}}{t_0 \star t_1' \in \dot{U}} \quad \text{(v)}
\end{align*}
\]

Lemma A.3. The following extra rules are derivable for lifted-totalised sets:

\[
\begin{align*}
\frac{U \subseteq \dot{U}}{} \quad \text{(i)} \quad \frac{\bot \in \dot{U}}{} \quad \text{(ii)} \quad \frac{t' \in \dot{U}}{} \quad \text{(iii)} \quad \frac{\neg \text{Pre} U t \quad t \in \dot{U}}{} \quad \text{(iv)} \quad \frac{\neg \text{Pre} U t_0 \quad t_0 \in \dot{U}}{} \quad \text{(v)}
\end{align*}
\]

Another relational completion lifts and totalises the relation, but is strict with respect to abortive behaviour: \( \bot \) maps only to \( \bot \).

Definition A.11 (Strict-lifted-totalisation).

\[
\tilde{U} = \{ z_0 \star z_1' \in T^\star \mid \text{Pre} U z_0 \Rightarrow z_0 \star z_1' \in U \land z_0 = \bot \Rightarrow z_1' = \bot' \}
\]

Our final relational completion is the abortive-lifted-totalisation: \( \bot \) and all before-states outside the precondition, map only to \( \bot \).

Definition A.12 (Abortive-lifted-totalisation).

\[
\hat{U} = \{ z_0 \star z_1' \in T^\star \mid z_0 \star z_1' \in U \lor (\neg \text{Pre} U z_0 \land z_1' = \bot') \}
\]