Refinement and the Z Schema Calculus

Lindsay Groves
School of Mathematical and Computing Sciences
Victoria University of Wellington
Wellington, New Zealand
lindsay@mcs.vuw.ac.nz

Abstract

It is well known that the principal operators in the Z schema calculus are not monotonic with respect to refinement, which limits their usefulness in software development. The usual reaction to this observation is to remove all schema operators before performing any kind of refinement, or to move to a different formalism such as B or the refinement calculus. This paper examines the interaction between refinement and the schema calculus more closely, showing exactly how non-monotonicity arises, and identifying various conditions under which components of schema expressions can be safely replaced by their refinements. This analysis uses an alternative characterisation of refinement, and a decomposition of the standard refinement relation into two simpler relations that allow us to study refinements that modify a specification in different ways.

1 Introduction

The literature on Z talks a lot about the schema calculus, and about refinement, but says very little about the interaction between them. It is well known, though not widely documented, that the schema calculus operators are not monotonic with respect to refinement,[1] which limits their usefulness in development beyond writing the initial specification. Most authors only discuss refinement on operations described as a single schema, and so appear to assume that schema operators are removed before performing any kind of refinement (e.g. [Dil90], [PST91], [DB01]); [WD96] is exceptional in discussing the interaction of refinement and promotion. Some authors suggest translating Z specifications into a procedural notion in a way that preserves some of the structure of the specifications (e.g. [Kin90], [Wor92], [Woo92], [CW98]), though this only works for a few special cases. Others conclude that Z should be replaced with a notation such as B or the refinement calculus, possibly extended with structuring facilities like those of Z (e.g. [War93], [Mah99]),[2] or modified so that the schema operators are monotonic (e.g. [HR00]).

This paper examines the interaction between refinement and the schema calculus more closely, explaining exactly why the schema operators are not monotonic and identifying some restricted cases where refinements can be performed on the components of a specification constructed using schema operations. This makes it possible to perform some refinements on the initial specification, without removing schema operators, before moving to a more procedural

[1] *Note to reviewers: I'm still looking for references where this is discussed — any suggestions would be most welcome!*

[2] Though, oddly, this is often done by preserving the non-monotonicity of the Z operators.

notation. In this paper, we only consider operation refinement, and focus on schema conjunction and disjunction since they are the most widely used schema operators and suffice to illustrate the kinds of properties we are interested in.

Section 2 introduces the essential elements of the schema calculus and some additional definitions. Section 3 gives some results about calculation of preconditions. Section 4 defines operation refinement for Z schemas, and introduces an alternative characterisation of refinement and a decomposition of the standard refinement relation into two simpler relations which are used to state some additional properties of operation refinement. Section 5 is the heart of the paper; it defines the notion of monotonicity and presents our results about monotonicity and non-monotonicity of schema conjunction and disjunction operators. Section 6 presents our conclusions, along with some comments about related and future work.

2 The Schema Calculus

We consider a subset of the schema calculus as defined in ZRM [Spi88], and we are primarily concerned with specifying operations that modify some state.

First, we have "simple" schemas, with the general form:

$$[\Delta S;\, x? : X;\, y! : Y \mid P]$$

where ΔS is an abbreviation for $S;\, S'$, denoting the sets of pre-state (S) and post-state (S') variables, $X?$ are the input variables and $y!$ are the output variables. Such a schema is interpreted as defining an operation that can exhibit various behaviours described by bindings for the variables in ΔS, $x?$ and $y!$ that satisfy P.

In most of what follows, we restrict our attention to operations that act only on the state and have no input or output — the results can all be adapted systematically to handle input and output, by treating inputs in the same way as pre-state variables and treating outputs in the same way as post-state variables. Thus, we consider schemas of the form:

$$[\Delta S \mid P]$$

We also have various ways of forming schema expressions. In this paper, we will only consider schema conjunction and disjunction.

The conjunction of two schemas is formed by merging their declarations (after normalising) and taking the conjunction of their predicates. Similarly, the disjunction of two schemas is formed by merging their declarations (after normalising) and taking the disjunction of their predicates. Thus, for normalised operation schemas, $A = [\Delta S \mid P]$ and $B = [\Delta S \mid Q]$, we have $A \wedge B = [\Delta S \mid P \wedge Q]$ and $A \vee B = [\Delta S \mid P \vee Q]$.[3]

We identify the following constant operations (for state S):

$$\mathit{False}_S = [\Delta S \mid \mathit{false}]$$

$$\mathit{True}_S = [\Delta S \mid \mathit{true}]$$

[3] We use $=$ to denote syntactic equality (including definitions) and $\equiv$ to denote semantic equivalence of schemas.

We also define $Vars_S(P)$ to be the variables declared in S that occur in P. We omit the S subscript when the state is clear from context.

The *precondition* of an operation in Z characterises those initial states (and inputs) for which the operation is defined. This is defined as follows.

Definition 2.1 (*Precondition*) For an operation $A = [\Delta S; x? : X; y! : Y \mid P]$,

$$\text{pre } A = \exists\, S', y! : Y \bullet A$$

♠

Ignoring inputs and outputs, this reduces to:

$$\text{pre } A \equiv \exists\, S' \bullet A$$

or $$\text{pre } A \equiv [S \mid \exists\, S' \bullet P]$$

Note that $A \Rightarrow \text{pre } A$ holds for all A.

Operationally, we can think of the precondition as characterising the initial states for which an implementation of the specification is required to terminate with a valid result. The precondition is thus sometimes referred to as the *domain of termination*.

We define some properties of operation specifications we'll need later.

Definition 2.2 For a specification $A = [\Delta S \mid P]$, we say:

(i) A is *total* iff $\text{pre } A \equiv True_S$ holds.

(ii) A is *satisfiable* iff $\text{pre } A \not\equiv False_S$ holds.

(iii) A is *deterministic* iff $(\forall\, S \mid \text{pre } A \bullet \exists_1\, S' \bullet A)$ holds. ♠

We also need the notion of two operations being *consistent*, which means that their conjunction is defined whenever they are both defined.

Definition 2.3 (*Consistency*) Two operations, A and B, on the same state, S, are *consistent* iff $\forall\, S \bullet \text{pre } A \wedge \text{pre } B \Rightarrow \text{pre } (A \wedge B)$ holds. ♠

Since $\text{pre } (A \wedge B) \Rightarrow \text{pre } A \wedge \text{pre } B$ always holds (see law 3.3 below), the added condition means that when A and B are consistent we have $\text{pre } (A \wedge B) \equiv \text{pre } A \wedge \text{pre } B$.[4]

3 Calculating Preconditions

In later sections, we will need to reason about preconditions for operations defined by schema expressions. Sometimes we can't do anything in terms of arbitrary schemas, so we have no option but to compute preconditions directly. In some cases, we can obtain weaker results that are useful for later proofs.

Throughout this section, we assume $A = [\Delta S \mid P]$ and $B = [\Delta S \mid Q]$.

[4]When there is no confusion, we often omit universal quantifiers surrounding formulas.

First, we note that taking the precondition of an operation that don't constrain the post-state variables leaves the operation unchanged. This is needed, for example, if we take the precondition of a precondition.

Law 3.1 pre $A \equiv A$ if $\mathit{Vars}(A) \cap S' = \varnothing$

Proof

$$
\begin{array}{ll}
 & \text{pre } A \\
\equiv & [\Delta S \mid \exists\, S' \bullet P] \\
\equiv & [\Delta S \mid P] \\
\equiv & A
\end{array}
\qquad \spadesuit
$$

3.1 Disjunction

Disjunction gives us a simple result ([Woo89], [WD96, p210], [Wor92, p125]).

Law 3.2 pre $(A \lor B) \equiv$ pre $A \lor$ pre B

Proof

$$
\begin{array}{ll}
 & \text{pre } (A \lor B) \\
\equiv & \text{pre } ([\Delta S \mid P] \lor [\Delta S \mid Q]) \\
\equiv & \text{pre } [\Delta S \mid P \lor Q] \\
\equiv & [\Delta S \mid \exists\, S' \bullet P \lor Q] \\
\equiv & [\Delta S \mid (\exists\, S' \bullet P) \lor (\exists\, S' \bullet Q)] \\
\equiv & [\Delta S \mid \exists\, S' \bullet P] \lor [\Delta S \mid \exists\, S' \bullet Q]) \\
\equiv & \text{pre } A \lor \text{pre } B
\end{array}
\qquad \spadesuit
$$

3.2 Conjunction

Conjunction doesn't work as well, because existential quantification does not distribute over (logical) conjunction. We do, however, get a weaker result which is useful later.

Law 3.3 pre $(A \land B) \Rightarrow$ pre $A \land$ pre B

Proof

$$
\begin{array}{ll}
 & \text{pre } (A \land B) \\
\equiv & \text{pre } ([\Delta S \mid P] \land [\Delta S \mid Q]) \\
\equiv & \text{pre } [\Delta S \mid P \land Q] \\
\equiv & [\Delta S \mid \exists\, S' \bullet P \land Q] \\
\Rightarrow & [\Delta S \mid (\exists\, S' \bullet P) \land (\exists\, S' \bullet Q)] \\
\equiv & [\Delta S \mid \exists\, S' \bullet P] \land [\Delta S \mid \exists\, S' \bullet Q] \\
\equiv & \text{pre } A \land \text{pre } B
\end{array}
\qquad \spadesuit
$$

We get a stronger result if P and Q have no variables in common.

Law 3.4 pre $(A \land B) \equiv$ pre $A \land$ pre B, if $\mathit{Vars}(A) \cap \mathit{Vars}(B) = \varnothing$.

Proof Same as for law 3.3, except that the implication step becomes an equivalence when P and Q have no variables in common. ♠

We get a similar result if one of the operations does not constrain its post-state variables. This will arise, for example, if it is the precondition of some operation.

Law 3.5 pre $(A \wedge B) \equiv A \wedge \text{pre } B$ if $Vars(P) \cap S' = \varnothing$

Proof

$$
\begin{array}{ll}
 & \text{pre } (A \wedge B) \\
\equiv & [\Delta S \mid \exists S' \bullet P \wedge Q] \\
\equiv & \quad \langle\ S' \text{ does not occur in } P\ \rangle \\
 & [\Delta S \mid P \wedge (\exists S' \bullet Q)] \\
\equiv & A \wedge \text{pre } B
\end{array}
$$

♠

4 Operation Refinement

An operation, A, is *refined* by another operation, B, if B can be used anywhere that A is required. This means that B must produce a valid result whenever A does, and any behaviour exhibited by B must be permitted by A. This is formalised as follows.

Definition 4.1 (*Refinement*) Let A and B be operations with state S, input $x? : X$ and output $y! : Y$, then B is an *operation refinement* of A, written $A \sqsubseteq B$, iff:

(i) $\forall S;\ x? : X \bullet \text{pre } A \Rightarrow \text{pre } B$ (Applicability)

(ii) $\forall \Delta S;\ x? : X;\ y! : Y \bullet \text{pre } A \wedge B \Rightarrow A$ (Correctness)

♠

This definition is essentially that given by Spivey [Spi88], and has been used by many other authors since (though some use turnstyle rather than implication). [WD96] and [DB01] instead give a semantic definition of refinement, based on a relational model ([DHR01] shows that this definition is equivalent to Spivey's), and [CW98] give a definition in terms of a weakest precondition semantics.[5] The names "applicability" and "correctness" for the conditions are from [PST91]; Wordsworth [Wor92] calls them "safety" and "liveness".

In this definition, inputs and outputs act in exactly the same way as initial and final state variables (observations), so in investigating properties of refinement, we can safely ignore them and just consider operations that only modify their state. We thus use the simplified conditions:

(i) $\forall S \bullet \text{pre } A \Rightarrow \text{pre } B$ (Applicability)

(ii) $\forall \Delta S \bullet \text{pre } A \wedge B \Rightarrow A$ (Correctness)

[5] Some authors use an alternative definition that does not allow weakening of the precondition; this corresponds to the notion of post-refinement introduced in Section 4.3.

Operation refinement can be understood as having two possible effects, which we often describe using more operational terminology:

- Refinement can increase the domain over which an operation is defined (i.e. weaken preconditions or increase termination).
- Refinement can reduce non-determinism (i.e. strengthen postconditions).

We will need to consider ways in which one operation fails to be a refinement of another (with the same state space). If operation B is not a refinement of A (on the same state space), then one of the conditions in definition 4.1 must fail to hold. That is, if $A \not\sqsubseteq B$, then either:

(i) $\neg\ \forall S \bullet \text{pre } A \Rightarrow \text{pre } B$

i.e. A is defined for some initial state for which B isn't, or

(ii) $\neg\ \forall \Delta S \bullet \text{pre } A \wedge B \Rightarrow A$

i.e. for some initial state where A is defined, B admits some behaviour that A doesn't.

4.1 Lattice properties of refinement

We also note the following lattice properties of operation refinement:

Law 4.1

(i) Operation refinement is a partial order on the set of all operations on a given state S; i.e. $\sqsubseteq$ is reflexive, transitive and anti-symmetric.

(ii) For any state, S, the operations over S form a meet semi-lattice; i.e. there is a bottom and a meet. The bottom is False_S, and the meet of operations A and B is $\text{pre } A \wedge \text{pre } B \wedge (A \vee B)$. (cf. [BEM92], which gives similar results in a relational setting.)

(iii) There is no top, and the join doesn't always exist. When it does exist, the join of operations A and B is given by $(\text{pre } A \vee \text{pre } B) \wedge (\text{pre } A \Rightarrow A) \wedge (\text{pre } B \Rightarrow B)$. This join exists provided that A and B are consistent.

A more general form of join operation, allowing data refinement of the components, is called *unification* in [BDBS99].

4.2 Decomposing the refinement relations

The effects described in definition 4.1 can be isolated into separate schemas, so that if $A \sqsubseteq B$, B can be written as $(A \wedge C) \vee D$, where C constrains the postcondition of A and has the same domain as A, and D augments the domain of A and has a domain disjoint from that of A. Moreover, B must be a refinement of A if it can be expressed in this way, provided that A and C are consistent.

Law 4.2 $A \sqsubseteq B$ iff B can be written as $(A \wedge C) \vee D$, where $\text{pre } A \equiv \text{pre } C$, $\text{pre } A \Rightarrow \neg \text{ pre } D$, and A and C are consistent.

The proof of law 4.2 requires the following lemmas.

Lemma 4.1 For any operations A and B, $A \Rightarrow B$ iff $A \equiv B \wedge C$ for some C where pre $A \equiv$ pre C and A and C are consistent. ♠

Lemma 4.2 For any operations A, B and C, pre $A \wedge B \Rightarrow C$ iff $B \equiv C \vee D$ for some D where pre $A \Rightarrow \neg$ pre C. ♠

Proof of law 4.2

($\Rightarrow$)

$$
\begin{array}{ll}
 & A \sqsubseteq B \\
\Rightarrow & \langle \text{ definition 4.1 } \rangle \\
 & \text{pre } A \wedge B \Rightarrow A \\
\equiv & \langle \text{ lemma 4.1 } \rangle \\
 & \text{pre } A \wedge B \equiv A \wedge C, \text{ where pre } A \equiv \text{pre } C \text{ and } A \text{ and } C \text{ are consistent} \\
\equiv & \langle \text{ lemma 4.2 } \rangle \\
 & B \equiv (A \wedge C) \vee D, \text{ where pre } A \equiv \text{pre } C, \text{ pre } A \Rightarrow \neg \text{ pre } D, \text{ and } A \text{ and } C \text{ are consistent}
\end{array}
$$

($\Leftarrow$) We now show that if $B \equiv (A \wedge C) \vee D$, where pre $A \equiv$ pre C, pre $A \Rightarrow \neg$ pre D, and A and C are consistent, the conditions of definition 4.1 are satisfied.

(i)

$$
\begin{array}{ll}
 & \text{pre } B \\
\equiv & \langle \text{ assumption, law 3.2 } \rangle \\
 & \text{pre } (A \wedge C) \vee \text{pre } D \\
\equiv & \langle\ A \text{ and } C \text{ are consistent, pre } A \equiv \text{pre } C\ \rangle \\
 & \text{pre } A \vee \text{pre } D \\
\Leftarrow & \langle \text{ logic } \rangle \\
 & \text{pre } A
\end{array}
$$

(ii)

$$
\begin{array}{ll}
 & \text{pre } A \wedge B \\
\equiv & \langle \text{ assumption } \rangle \\
 & \text{pre } A \wedge ((A \wedge C) \vee D) \\
\equiv & \langle \text{ logic } \rangle \\
 & (\text{pre } A \wedge A \wedge C) \vee (\text{pre } A \wedge D) \\
\equiv & \langle\ A \Rightarrow \text{pre } A, \text{ pre } A \Rightarrow \neg \text{ pre } D\ \rangle \\
 & A \wedge C \\
\Rightarrow & \langle \text{ logic } \rangle \\
 & A
\end{array}
$$

♠

4.3 Pre- and post-refinement

In later discussion it will be useful to distinguish refinements that only modify an operation in one of the ways described in definition 4.1.

Definition 4.2 (*Pre-refinement*) A is *pre-refined* by B, written $A \sqsubseteq_{pre} B$, iff B refines A, but does not reduce non-determinism for any initial state in the precondition of A; i.e. $A \sqsubseteq_{pre} B$ iff $A \sqsubseteq B$ and $\forall \Delta S \bullet A \Rightarrow B$. ♠

Definition 4.3 (*Post-refinement*) A is *post-refined* by B, written $A \sqsubseteq_{post} B$, iff B refines A, but is only defined for initial states where A is defined; i.e. $A \sqsubseteq_{post} B$ iff $A \sqsubseteq B$ and $\forall S \bullet \text{pre } B \Rightarrow \text{pre } A$. ♠

We obtain the following properties of $\sqsubseteq_{pre}$ and $\sqsubseteq_{post}$ by simplifying their definitions.

Law 4.3 $A \sqsubseteq_{pre} B$ iff $\forall \Delta S \bullet A \equiv \text{pre } A \wedge B$.

Proof

$$
\begin{array}{ll}
& A \sqsubseteq_{pre} B \\
\equiv & \langle \text{ definitions 4.2 and 4.1 } \rangle \\
& (\forall S \bullet \text{pre } A \Rightarrow \text{pre } B) \wedge (\forall \Delta S \bullet \text{pre } A \wedge B \Rightarrow A) \wedge (\forall \Delta S \bullet A \Rightarrow B) \\
\equiv & \langle \text{ logic, } S' \text{ not free in pre } A \Rightarrow \text{pre } B \ \rangle \\
& (\forall \Delta S \bullet (\text{pre } A \Rightarrow \text{pre } B) \wedge (\text{pre } A \wedge B \Rightarrow A) \wedge (A \Rightarrow B)) \\
\equiv & \langle \text{ logic, } A \Rightarrow \text{pre } A \ \rangle \\
& \forall \Delta S \bullet A \equiv \text{pre } A \wedge B
\end{array}
$$
♠

Law 4.4 $A \sqsubseteq_{post} B$ iff $\forall S \bullet \text{pre } A \equiv \text{pre } B$ and $\forall \Delta S \bullet B \Rightarrow A$.

Proof

$$
\begin{array}{ll}
& A \sqsubseteq_{post} B \\
\equiv & \langle \text{ definitions 4.3 and 4.1 } \rangle \\
& (\forall S \bullet \text{pre } A \Rightarrow \text{pre } B) \wedge (\forall \Delta S \bullet \text{pre } A \wedge B \Rightarrow A) \wedge (\forall S \bullet \text{pre } B \Rightarrow \text{pre } A) \\
\equiv & \langle \text{ logic } \rangle \\
& (\forall S \bullet \text{pre } A \equiv \text{pre } B) \wedge (\forall \Delta S \bullet \text{pre } A \wedge B \Rightarrow A) \\
\equiv & \langle \text{ pre } A \equiv \text{pre } B,\ B \Rightarrow \text{pre } B \ \rangle \\
& (\forall S \bullet \text{pre } A \equiv \text{pre } B) \wedge (\forall \Delta S \bullet B \Rightarrow A)
\end{array}
$$
♠

The next two laws show how pre- and post-refinements can be expressed as special cases of the form described in law 4.2.

Law 4.5 $A \sqsubseteq_{pre} B$ iff B can be written as $A \vee C$, where $\text{pre } A \Rightarrow \neg \text{ pre } C$.

Proof

$$
\begin{aligned}
& A \sqsubseteq_{pre} B \\
\equiv \quad & \langle \text{ law 4.3 } \rangle \\
& \text{pre } A \wedge B \equiv A \\
\equiv \quad & \langle \text{ lemma 4.2 } \rangle \\
& B \equiv A \vee C, \text{ where pre } A \Rightarrow \neg \text{ pre } C
\end{aligned}
$$

♠

Law 4.6 $A \sqsubseteq_{post} B$ iff B can be written as $A \wedge C$, where pre $A \equiv$ pre C, and A and C are consistent.

Proof

$$
\begin{aligned}
& A \sqsubseteq_{post} B \\
\equiv \quad & \langle \text{ law 4.4 } \rangle \\
& (\text{pre } A \equiv \text{pre } A) \wedge (B \Rightarrow A) \\
\equiv \quad & \langle \text{ lemma 4.1 } \rangle \\
& B \equiv A \wedge C, \text{ where pre } A \equiv \text{pre } C \text{ and } A \text{ and } C \text{ are consistent}
\end{aligned}
$$

♠

The next two laws reveal relationships between $\sqsubseteq_{pre}$ and $\sqsubseteq_{post}$.

If A is both pre-refined and post-refined by B, then B cannot alter either the precondition or postcondition of A, so must be equivalent to A.

Law 4.7 $A \sqsubseteq_{pre} B$ and $A \sqsubseteq_{post} B$ iff $A \equiv B$.

Proof

$$
\begin{aligned}
& (A \sqsubseteq_{pre} B) \wedge (A \sqsubseteq_{post} B) \\
\equiv \quad & \langle \text{ laws 4.3 and 4.4 } \rangle \\
& (\forall \Delta S \bullet A \equiv \text{pre } A \wedge B) \wedge (\forall S \bullet \text{pre } A \equiv \text{pre } B) \wedge (\forall \Delta S \bullet \text{pre } A \wedge B \Rightarrow A) \\
\equiv \quad & \langle \text{ logic, } B \Rightarrow \text{pre } B \text{ } \rangle \\
& (\forall \Delta S \bullet A \equiv B)
\end{aligned}
$$

♠

The next two laws show that any operation refinement can be obtained as a post-refinement followed by a pre-refinement, or as a pre-refinement followed by a post-refinement.

Law 4.8 $A \sqsubseteq B$ iff $A \sqsubseteq_{post} E$ and $E \sqsubseteq_{pre} B$, for some E.

Proof

$$
\begin{array}{ll}
& A \sqsubseteq B \\
\equiv & \langle \text{ law 4.2 } \rangle \\
& B \equiv (A \wedge C) \vee D, \text{ where pre } A \equiv \text{pre } C, \text{ pre } A \Rightarrow \neg \text{ pre } D, \text{ and } A \text{ and } C \text{ are consistent} \\
\equiv & \left\langle \begin{array}{l} \text{laws 4.6 and 4.5, pre } (A \wedge C) \equiv \text{pre } A, \text{ since } A \text{ and } C \text{ are consistent and} \\ \text{pre } A \equiv \text{pre } C \end{array} \right\rangle \\
& (A \sqsubseteq_{post} A \wedge C) \wedge (A \wedge C \sqsubseteq_{pre} (A \wedge C) \vee D), \text{ with } A \text{ and } C \text{ as above} \\
\equiv & \langle\ B \equiv (A \wedge C) \vee D\ \rangle \\
& (A \sqsubseteq_{post} A \wedge C) \wedge (A \wedge C \sqsubseteq_{pre} B)
\end{array}
$$

So the result follows by taking $E = A \wedge C$. ♠

Law 4.9 $A \sqsubseteq B$ iff $A \sqsubseteq_{pre} E$ and $E \sqsubseteq_{post} B$, for some E.

Proof

$$
\begin{array}{ll}
& A \sqsubseteq B \\
\equiv & \langle \text{ law 4.2 } \rangle \\
& B \equiv (A \wedge C) \vee D, \text{ where pre } A \equiv \text{pre } C, \text{ pre } A \Rightarrow \neg \text{ pre } D, \text{ and } A \text{ and } C \text{ are consistent} \\
\equiv & \langle \text{ logic } \rangle \\
& B \equiv (A \vee D) \wedge (C \vee D), \text{ with } A \text{ and } C \text{ as above} \\
\equiv & \left\langle \begin{array}{l} \text{laws 4.5 and 4.6, pre } (A \vee D) \equiv \text{pre } (C \vee D), \text{ since pre } A \equiv \text{pre } C, \\ \text{and } A \vee D \text{ and } C \vee D \text{ are consistent, since } A \text{ and } C \text{ are consistent} \end{array} \right\rangle \\
& (A \sqsubseteq_{pre} A \vee D) \wedge (A \vee D \sqsubseteq_{post} (A \vee D) \wedge (C \vee D)), \text{ with } A \text{ and } C \text{ as above} \\
\equiv & \langle\ B \equiv (A \vee D) \wedge (C \vee D)\ \rangle \\
& (A \sqsubseteq_{post} A \wedge C) \wedge (A \wedge C \sqsubseteq_{pre} B)
\end{array}
$$

So the result follows by taking $E = A \vee D$. ♠

5 Monotonicity

A schema operator is monotonic with respect to refinement in a given argument position if replacing that argument by a refinement gives a refinement of the entire operation.

Definition 5.1 (*Monotonicity*) An n-ary schema operator $\mathcal{F}$ is *monotonic with respect to refinement* in its kth argument, for $1 \leq k \leq n$, if $S \sqsubseteq S'$ implies $\mathcal{F}(a_1, \cdots, a_{k-1}, S, a_{k+1}, \cdots, a_n) \sqsubseteq \mathcal{F}(a_1, \cdots, a_{k-1}, S', a_{k+1}, \cdots, a_n)$. ♠

This is an important property because, when it holds, it means that components of a composite specification can be refined independently of the rest of the specification. Unfortunately, many of the Z schema operators are not monotonic, which limits their usefulness in software development.

We now consider monotonicity and non-monotonicity of the principal Z schema operators. Where the operators are not monotonic, we explore conditions under which component replacement is safe. Our aim is to find rules that might be useful in practice, so we are interested in finding a range of conditions that make the operators monotonic, rather than just finding the most general conditions.

We only consider conjunction and disjunction in detail, as they appear to be the operators most widely used in structuring specifications and suffice to illustrate the kinds of properties we are interested in; we discuss the others briefly in Section 6. Since conjunction and disjunction are commutative, we don't need to consider the two argument positions separately.

5.1 Conjunction

It is easy to see that schema conjunction is not monotonic with respect to refinement, because reducing non-determinism in its components can leave the operation with no choices at all.

Example 5.1 Consider the operation $Op_{5.1} = A \wedge B$, where:

$$A = [x, x' : \mathbb{N} \mid x' \in \{0, 1\}]$$
$$B = [x, x' : \mathbb{N} \mid x' \in \{1, 2\}]$$

Both A and B are non-deterministic, and can be refined by reducing non-determinism. For example, A is refined by:

$$A1 = [x, x' : \mathbb{N} \mid x' = 0]$$

since $x' = 0 \Rightarrow x' \in \{0, 1\}$, and B is refined by:

$$B1 = [x, x' : \mathbb{N} \mid x' = 2]$$

since $x' = 2 \Rightarrow x' \in \{1, 2\}$. But $A \wedge B \not\sqsubseteq A1 \wedge B1$, since we have:

$$\begin{array}{ll} & A \wedge B \\ \equiv & [x, x' : \mathbb{N} \mid x' \in \{0, 1\}] \wedge [x, x' : \mathbb{N} \mid x' \in \{1, 2\}] \\ \equiv & [x, x' : \mathbb{N} \mid x' \in \{0, 1\} \wedge x' \in \{1, 2\}] \\ \equiv & [x, x' : \mathbb{N} \mid x' = 1] \end{array}$$

and:

$$\begin{array}{ll} & A1 \wedge B1 \\ \equiv & [x, x' : \mathbb{N} \mid x' = 0] \wedge [x, x' : \mathbb{N} \mid x' = 2] \\ \equiv & [x, x' : \mathbb{N} \mid x' = 0 \wedge x' = 2] \\ \equiv & [x, x' : \mathbb{N} \mid \mathit{false}] \end{array}$$

And $x' = 1 \not\Rightarrow \mathit{false}$.

Thus, although we have $A \sqsubseteq A1$ and $B \sqsubseteq B1$, it is not safe to replace A by $A1$, or B by $B1$, in $Op_{5.1}$. ♠

Having seen how non-monotonicity can arise, we will now explore some restricted cases in which conjunction is monotonic, and thus identify conditions under which it is safe to replace a component of a conjunction by a refinement.

First, we observe that reducing non-determinism is the *only* way in which refining the components of a conjunction can lead to non-monotonicity. Conversely, increasing termination of the components cannot reduce non-determinism in a conjunction, and cannot lead to non-monotonicity. Thus, we can safely refine the components of a disjunction in a way that only increases termination; i.e. is a pre-refinement.

Law 5.1 If $A \sqsubseteq_{pre} A1$ and $B \sqsubseteq_{pre} B1$, then $A \wedge B \sqsubseteq_{pre} A1 \wedge B1$.

Proof When $A \sqsubseteq_{pre} A1$ and $B \sqsubseteq_{pre} B1$, we can assume (by law 4.5) that there are operations C and D such that $A1 = A \vee C$ and $B1 = B \vee D$, where $\text{pre } A \Rightarrow \neg \text{ pre } C$, $\text{pre } B \Rightarrow \neg \text{ pre } D$ (or, equivalently, $\text{pre } C \Rightarrow \neg \text{ pre } A$ and $\text{pre } D \Rightarrow \neg \text{ pre } B$).

We can now write $A1 \wedge B1$ in the form required for law 4.5:

$$\begin{array}{ll} & A1 \wedge B1 \\ \equiv & (A \vee C) \wedge (B \vee D) \\ \equiv & (A \wedge B) \vee ((A \wedge D) \vee (C \wedge B) \vee (C \wedge D)) \end{array}$$

We can also show that the domains of $A \wedge B$ and $(A \wedge D) \vee (C \wedge B) \vee (C \wedge D)$ are disjoint.

$$\begin{array}{ll} & \text{pre } ((A \wedge D) \vee (C \wedge B) \vee (C \wedge D)) \\ \equiv & \quad \langle \text{ law 3.2 } \rangle \\ & \text{pre } (A \wedge D) \vee \text{pre } (C \wedge B) \vee \text{pre } (C \wedge D)) \\ \Rightarrow & \quad \langle \text{ law 3.3 } \rangle \\ & (\text{pre } A \wedge \text{pre } D) \vee (\text{pre } C \wedge \text{pre } B) \vee (\text{pre } C \wedge \text{pre } D) \\ \Rightarrow & \quad \langle \text{ pre } C \Rightarrow \neg \text{ pre } A \text{ and pre } D \Rightarrow \neg \text{ pre } B \rangle \\ & (\text{pre } A \wedge \neg \text{ pre } B) \vee (\neg \text{ pre } A \wedge \text{pre } B) \vee (\neg \text{ pre } A \wedge \neg \text{ pre } B) \\ \equiv & \quad \langle \text{ logic } \rangle \\ & \neg \text{ pre } A \vee \neg \text{ pre } B \\ \Rightarrow & \quad \langle \text{ contrapositive of law 3.3 } \rangle \\ & \neg \text{ pre } (A \wedge B) \end{array}$$

Thus, by law 4.5, we have $A \wedge B \sqsubseteq_{pre} A1 \wedge B1$. ♠

Example 5.2 Consider the operation $Op_{5.2} = A \wedge B$, where:

$$A = [x, x' : \mathbb{Z} \mid even(x) \wedge even(x')]$$
$$B = [x, x' : \mathbb{N}_1 \mid x \leq x' \leq 2 * x]$$

We can refine both A and B by weakening their preconditions:

$$A1 = [x, x' : \mathbb{Z} \mid even(x) \equiv even(x')]$$

$$B1 = [x, x' : \mathbb{Z} \mid abs(x) \leq x' \leq 2 * abs(x)]$$

Since these refinements only modify the preconditions of A and B, by law 5.1, we can replace A by $A1$ and B by $B1$ in $Op_{5.2}$, i.e. $A \land B \sqsubseteq A1 \land B1$. ♠

We can get more general results by observing that decreasing non-determinism in the components of a conjunction doesn't necessarily lead to non-monotonicity, and is safe provided that it does not remove all possible behaviours. We can identify several conditions under which this can be guaranteed.

First, we observe that refining the components of a conjunction cannot lead to incorrectness. This means that to show $A \land B \sqsubseteq A1 \land B1$, given $A \sqsubseteq A1$ and $B \sqsubseteq B1$, we only need to consider the applicability condition of definition 4.1, which can be specialised as shown in the following law.

Law 5.2 $A \land B \sqsubseteq A1 \land B1$ if $A \sqsubseteq A1$, $B \sqsubseteq B1$, and $A1$ and $B1$ are consistent whenever A and B are.

Proof We prove that $A \land B \sqsubseteq A1 \land B1$ using definition 4.1.

Applicability: The assumption that $A1$ and $B1$ are consistent whenever A and B are consistent means that $\forall S \bullet \text{pre}\,(A \land B) \Rightarrow \text{pre}\,(A1 \land B1)$ holds.

Correctness:

$$\begin{array}{ll} & \text{pre}\,(A \land B) \land A1 \land B1 \\ \Rightarrow & \langle \text{ law 3.3 } \rangle \\ & \text{pre}\,A \land \text{pre}\,B \land A1 \land B1 \\ \Rightarrow & \langle\, A \sqsubseteq A1 \text{ and } B \sqsubseteq B1, \text{ definition 4.1 } \rangle \\ & A \land B \end{array}$$

Thus, by definition 4.1, we have $A \land B \sqsubseteq A1 \land B1$. ♠

Example 5.3 Consider the operation $Op_{5.3} = A \land B$, where:

$$A = [x, x' : \mathbb{N} \mid x' \in \{0, 1, 2\}]$$

$$B = [x, x' : \mathbb{N} \mid x' \in \{1, 2, 3\}]$$

We can refine both A and B by strengthening their postconditions:

$$A1 = [x, x' : \mathbb{N} \mid x' \in \{0, 1\}]$$

$$B1 = [x, x' : \mathbb{N} \mid x' \in \{1, 2\}]$$

Now, $A1$ and $B1$ are always consistent, as are A and B, so by law 5.2 we can replace A by $A1$ and B by $B1$ in $Op_{5.3}$, i.e. $A \land B \sqsubseteq A1 \land B1$. ♠

Finally, we observe that the components of a conjunction can be refined safely if they (and their refinements) act on disjoint parts of the state.

Law 5.3 If $A \sqsubseteq A1$ and $B \sqsubseteq B1$, where $A = [\Delta S \mid P]$, $B = [\Delta S \mid Q]$, $A1 = [\Delta S \mid P1]$ and $B1 = [\Delta S \mid Q1]$, and $Vars(P) \cap Var(Q) = \varnothing$, $Vars(P1) \subseteq Vars(P)$ and $Vars(Q1) \subseteq Vars(Q)$, then $A \wedge B \sqsubseteq A1 \wedge B1$.

Proof With these assumptions, law 3.4 gives us pre $(A \wedge B) \equiv$ pre $A \wedge$ pre B and pre $(A1 \wedge B1) \equiv$ pre $A1 \wedge$ pre $B1$, so the conditions of law 5.2 are satisfied. ♠

We can apply law 5.3 to operations that act on disjoint states by first combining their states; i.e. given $A = [\Delta S \mid P]$, $B = [\Delta T \mid Q]$, $A1 = [\Delta S \mid P1]$ and $B1 = [\Delta T \mid Q1]$, where $S \cap T = \varnothing$, we have $A \wedge B \equiv [\Delta S;\ T \mid P \wedge Q] \equiv [\Delta S;\ T \mid P] \wedge [\Delta S;\ T \mid Q]$ and $A1 \wedge B1 \equiv [\Delta S;\ T \mid P1 \wedge Q1] \equiv [\Delta S;\ T \mid P1] \wedge [\Delta S;\ T \mid Q1]$.

Example 5.4 Consider the operation $Op_{5.4} = A \wedge B$, where:

$$A = [x, x' : \mathbb{N} \mid x < 100 \wedge x' > x]$$
$$B = [y, y' : \mathbb{N} \mid y < 10 \wedge y' \mathsf{\ mod\ } 10 = y]$$

We can refine both A and B by weakening their preconditions and/or strengthening their postconditions:

$$A1 = [x, x' : \mathbb{N} \mid x' = 2 * x + 1]$$
$$B1 = [y, y' : \mathbb{N} \mid y < 10 \wedge y' = y + 400]$$

Since A and B, and $A1$ and $B1$, operate on disjoint states, we can replace A by $A1$ and B by $B1$ in $Op_{5.4}$, i.e. $A \wedge B \sqsubseteq A1 \wedge B1$. ♠

The reason why conjunction is not monotonic is that Z does not distinguish between undefinedness (non-termination) and infeasibility. This decision is defensible when one is only concerned with writing specifications, but creates problems when we consider refinement. One way to avoid the problem is to separate preconditions from postconditions, as in the refinement calculus ([BvW98, Mor94b]). In this case, we can define the conjunction of specifications $A = [\Delta S \mid PreA \mid PostA]$ and $B = [\Delta S \mid PreB \mid PostB]$ to be $[\Delta S \mid PreA \wedge PreB \mid PostA \wedge PostB]$, which is $[\Delta S \mid false \mid PostA \wedge PostB]$ if the preconditions of A and B are disjoint and $[\Delta S \mid PreA \wedge PreB \mid false]$ if the postconditions are inconsistent. We can now interpret $[\Delta S \mid false \mid PostA \wedge PostB]$ as a specification that never requires a result to be produced (**abort** in the refinement calculus), and $[\Delta S \mid PreA \wedge PreB \mid false]$ as a specification that is never able to produce a result, even when required to do so, which is a top element in the refinement lattice (**magic** in the refinement calculus). This approach is discussed in [HR00].

5.2 Disjunction

Schema disjunction is not monotonic with respect to refinement because increasing definedness in the components can admit previously excluded behaviours. This occurs when weakening the precondition of one operation adds behaviours for initial states in the precondition of the other operation.

Example 5.5 Consider the operation $Op_{5.5} = A \vee B$, where:

$$A = [x, x' : \mathbb{N} \mid x = 0 \wedge x' = 0]$$
$$B = [x, x' : \mathbb{N} \mid x = 1 \wedge x' = 1]$$

Neither A nor B is total, so they can both be refined by weakening their preconditions. For example, A is refined by:

$$A1 \;=\; [x, x' : \mathbb{N} \mid x' = 0]$$

since $x = 0 \Rightarrow \mathit{true}$, and B is refined by:

$$B1 \;=\; [x, x' : \mathbb{N} \mid x' = 1]$$

since $x = 1 \Rightarrow \mathit{true}$. But $A \vee B \not\sqsubseteq A1 \vee B1$, since we have:

$$\begin{array}{ll} & A \vee B \\ \equiv & [x, x' : \mathbb{N} \mid x = 0 \wedge x' = 0] \vee [x, x' : \mathbb{N} \mid x = 1 \wedge x' = 1] \\ \equiv & [x, x' : \mathbb{N} \mid x = 0 \wedge x' = 0 \vee x = 1 \wedge x' = 1] \end{array}$$

and:

$$\begin{array}{ll} & A1 \vee B1 \\ \equiv & [x, x' : \mathbb{N} \mid x' = 0] \vee [x, x' : \mathbb{N} \mid x' = 1] \\ \equiv & [x, x' : \mathbb{N} \mid x' = 0 \vee x' = 1] \end{array}$$

So $A1 \vee B1$ admits the behaviour $x = 0 \wedge x' = 1$, which is not permitted by $A \vee B$.

Thus, although we have $A \sqsubseteq A1$ and $B \sqsubseteq B1$, it is not safe to replace A by $A1$, or B by $B1$, in $Op_{5.5}$. ♠

Having seen how non-monotonicity can arise, we will now explore some restricted cases in which disjunction is monotonic, and thus identify conditions under which it is safe to replace a component of a disjunction by a refinement.

First, we observe that increasing termination is the *only* way in which refining the components of a disjunction can lead to non-monotonicity. Conversely, decreasing non-determinism in the components cannot increase termination in a disjunction, and cannot lead to non-monotonicity. Thus, we can safely refine the components of a disjunction in a way that only decreases non-determinism; i.e. is a post-refinement.

Law 5.4 If $A \sqsubseteq_{post} A1$ and $B \sqsubseteq_{post} B1$, then $A \vee B \sqsubseteq_{post} A1 \vee B1$.

Proof We prove that $A \vee B \sqsubseteq_{post} A1 \vee B1$ using law 4.4.

(i) $\text{pre}\,(A \vee B) \equiv \text{pre}\,(A1 \vee B1)$, since $\text{pre}\,A \equiv \text{pre}\,A1$ and $\text{pre}\,B \equiv \text{pre}\,B1$, by law 4.4.

(ii) $A1 \vee B1 \Rightarrow A \vee B$, since $A1 \Rightarrow A$ and $B1 \Rightarrow B$, by law 4.4.

Thus, by law 4.4, we have $A \wedge B \sqsubseteq_{post} A1 \wedge B1$. ♠

Example 5.6 Consider the operation $Op_{5.6} = A \vee B$, where:

$$A = [x, x' : \mathbb{Z} \mid x \geq 0 \wedge x' \geq x]$$
$$B = [x, x' : \mathbb{Z} \mid x \leq 0 \wedge x' \leq x]$$

We can refine both A and B by strengthening their postconditions:

$$A1 = [x, x' : \mathbb{Z} \mid x \geq 0 \wedge x' \geq x \wedge even(x')]$$
$$B1 = [x, x' : \mathbb{Z} \mid x \leq 0 \wedge x' \leq x \wedge odd(x')]$$

Since these refinements only modify the postconditions of A and B, by law 5.4, we can replace A by $A1$ and B by $B1$ in $Op_{5.6}$, i.e. $A \vee B \sqsubseteq A1 \vee B1$. ♠

We can get more general results by observing that increasing termination in the components of a disjunction doesn't necessarily lead to non-monotonicity, and is safe provided that it does not admit previously excluded behaviours. We can identify several conditions under which this can be guaranteed.

First, we observe that refining the components of a disjunction cannot lead to undefinedness. This means that to show $A \vee B \sqsubseteq A1 \vee B1$, given $A \sqsubseteq A1$ and $B \sqsubseteq B1$, we only need to consider the correctness condition of definition 4.1, which can be specialised as shown in the following law.

Law 5.5 $A \vee B \sqsubseteq A1 \vee B1$ if $A \sqsubseteq A1$, $B \sqsubseteq B1$, pre $A \wedge B1 \Rightarrow A \vee B$ and pre $B \wedge A1 \Rightarrow A \vee B$.

Proof To show that $A \vee B \sqsubseteq A1 \vee B1$, using definition 4.1, we need to show that the two conditions in definition 4.1 hold.

The applicability condition always holds; i.e. if $A \sqsubseteq A1$ and $B \sqsubseteq B1$, then pre $(A \vee B) \Rightarrow$ pre $(A1 \vee B1)$.

$$
\begin{array}{ll}
 & \text{pre } (A \vee B) \\
\equiv & \text{pre } A \vee \text{pre } B \\
\Rightarrow & \text{pre } A1 \vee \text{pre } B1 \\
\equiv & \text{pre } (A1 \vee B1)
\end{array}
$$

Thus, we only need to consider the correctness condition, which can be simplified as follows:

$$
\begin{array}{ll}
 & \text{pre } (A \vee B) \wedge (A1 \vee B1) \Rightarrow A \vee B \\
\equiv & \quad \langle \text{ law 3.2, logic } \rangle \\
 & (\text{pre } A \wedge A1) \vee (\text{pre } A \wedge B1) \vee (\text{pre } B \wedge A1) \vee (\text{pre } B \wedge B1) \Rightarrow A \vee B \\
\Leftarrow & \quad \langle\ A \sqsubseteq A1,\ B \sqsubseteq B1, \text{ definition 4.1 } \rangle \\
 & (\text{pre } A \wedge B1) \vee (\text{pre } B \wedge A1) \Rightarrow A \vee B \\
\equiv & \quad \langle \text{ logic } \rangle \\
 & (\text{pre } A \wedge B1 \Rightarrow A \vee B) \wedge (\text{pre } B \wedge A1 \Rightarrow A \vee B)
\end{array}
$$
♠

While law 5.5 provides a general way of showing that refining the components of a disjunction is safe, it is more complex than we may need in practice. We therefore now investigate some simpler conditions where refining the components of a disjunction can be shown to be safe.

Refining the components of a disjunction will not admit previously excluded behaviours if the refined components have disjoint preconditions. In this case the preconditions of the original components must also be disjoint.

Law 5.6 If $A \sqsubseteq A1$, $B \sqsubseteq B1$ and $\mathrm{pre}\ A1 \Rightarrow \neg\ \mathrm{pre}\ B1$, then $A \vee B \sqsubseteq A1 \vee B1$.

Proof

$$
\begin{array}{ll}
 & \mathrm{pre}\ A \wedge B1 \\
\Rightarrow & \quad \langle\ \mathrm{pre}\ A \Rightarrow \mathrm{pre}\ A1 \text{ and } B1 \Rightarrow \mathrm{pre}\ B1\ \rangle \\
 & \mathrm{pre}\ A1 \wedge \mathrm{pre}\ B1 \\
\Rightarrow & \quad \langle\ \mathrm{pre}\ A1 \Rightarrow \neg\ \mathrm{pre}\ B1\ \rangle \\
 & \mathit{False} \\
\Rightarrow & \quad \langle\ \text{logic}\ \rangle \\
 & A \vee B
\end{array}
$$

Similarly, we get $\mathrm{pre}\ B \wedge A1 \Rightarrow A \vee B$, so by law 5.5, we have $A \vee B \sqsubseteq A1 \vee B1$. ♠

Example 5.7 Consider the operation $Op_{5.7} = A \vee B$, where:

$$
\begin{array}{lcl}
A & = & [x, x' : \mathbb{Z} \mid x \in \{-1, -2\} \wedge x' = -1] \\
B & = & [x, x' : \mathbb{Z} \mid x \in \{1, 2\} \wedge x' = 1]
\end{array}
$$

We now refine A and B by weakening their preconditions in a way that keeps them disjoint:

$$
\begin{array}{lcl}
A1 & = & [x, x' : \mathbb{Z} \mid x < 0 \wedge x' = -1] \\
B1 & = & [x, x' : \mathbb{Z} \mid x > 0 \wedge x' = 1]
\end{array}
$$

Now, we have $\mathrm{pre}\ A1 \equiv x < 0 \Rightarrow \neg\ (x > 0) \equiv \neg\ \mathrm{pre}\ B1$. Thus, by law 5.6, we can replace A by $A1$ and B by $B1$ in $Op_{5.7}$, i.e. $A \vee B \sqsubseteq A1 \vee B1$. ♠

A special case of law 5.6 arises when the components of the disjunction (and their refinements) act on disjoint parts of the state. This situation is unlikely to arise in practice, because whichever disjunct is chosen, the variables in the other disjunct are completely unconstrained.

Law 5.7 If $A \sqsubseteq A1$ and $B \sqsubseteq B1$, where $A = [\Delta S \mid P]$, $B = [\Delta S \mid Q]$, $A1 = [\Delta S \mid P1]$ and $B1 = [\Delta S \mid Q1]$, and $\mathit{Vars}(P) \cap \mathit{Var}(Q) = \varnothing$, $\mathit{Vars}(P1) \subseteq \mathit{Vars}(P)$ and $\mathit{Vars}(Q1) \subseteq \mathit{Vars}(Q)$, then $A \vee B \sqsubseteq A1 \vee B1$.

Proof With these assumptions, the conditions of law 5.9 are satisfied. ♠

We can apply law 5.7 to operations that act on disjoint states by first combining their states; i.e. given $A = [\Delta S \mid P]$, $B = [\Delta T \mid Q]$, $A1 = [\Delta S \mid P1]$ and $B1 = [\Delta T \mid Q1]$, where $S \cap T = \varnothing$, we have $A \vee B \equiv [\Delta S;\ T \mid P \vee Q] \equiv [\Delta S;\ T \mid P] \vee [\Delta S;\ T \mid Q]$ and $A1 \vee B1 \equiv [\Delta S;\ T \mid P1 \vee Q1] \equiv [\Delta S;\ T \mid P1] \vee [\Delta S;\ T \mid Q1]$.

Example 5.8 Consider the operation $Op_{5.8} = A \vee B$, where:

$$\begin{aligned} A &= [x, x' : \mathbb{Z} \mid x < hi \wedge x < x' \leq hi] \\ B &= [y, y' : \mathbb{Z} \mid y > lo \wedge y > y' \geq lo] \end{aligned}$$

We can refine both A and B by weakening their preconditions and/or strengthening their postconditions:

$$\begin{aligned} A1 &= [x, x' : \mathbb{Z} \mid x' = x + 1] \\ B1 &= [x, x' : \mathbb{Z} \mid y' = y - 1] \end{aligned}$$

Since A and B, and $A1$ and $B1$, operate on disjoint states, by law 5.7, we can replace A by $A1$ and B by $B1$ in $Op_{5.8}$, i.e. $A \vee B \sqsubseteq A1 \vee B1$. ♠

We can also guarantee that refining the components of a disjunction will not admit previously excluded behaviours if the refinement of one component is only defined outside the precondition of the other (original) component, and vice versa. This also requires the preconditions of the original components to be disjoint, but does allow the preconditions of the refinements to overlap.

Law 5.8 If $A \sqsubseteq A1$, $B \sqsubseteq B1$, pre $A \Rightarrow \neg$ pre $B1$ and pre $B \Rightarrow \neg$ pre $A1$, then $A \vee B \sqsubseteq A1 \vee B1$.

Proof

$$\begin{aligned} & \text{pre } A \wedge B1 \\ \Rightarrow \quad & \langle \text{ pre } A \Rightarrow \text{pre } B1 \text{ and } B1 \Rightarrow \text{pre } B1 \rangle \\ & \mathit{False} \\ \Rightarrow \quad & \langle \text{ logic } \rangle \\ & A \vee B \end{aligned}$$

Similarly, we get pre $B \wedge A1 \Rightarrow A \vee B$, so by law 5.5, we have $A \vee B \sqsubseteq A1 \vee B1$. ♠

Example 5.9 Consider the operation $Op_{5.9} = A \vee B$, where:

$$\begin{aligned} A &= [x, x' : \mathbb{Z} \mid x \in \{-1, -2\} \wedge x' = -1] \\ B &= [x, x' : \mathbb{Z} \mid x \in \{1, 2\} \wedge x' = 1] \end{aligned}$$

We now refine A and B by weakening their preconditions is a way that does not overlap with the precondition of the other (original) component.

$$\begin{aligned} A1 &= [x, x' : \mathbb{Z} \mid x \leq 0 \wedge x' = -1] \\ B1 &= [x, x' : \mathbb{Z} \mid x \geq 0 \wedge x' = 1] \end{aligned}$$

Now, we have pre $A \equiv x \in \{-1, -2\} \Rightarrow \neg\,(x \geq 0) \equiv \neg$ pre $B1$, and pre $B \equiv x \in \{1, 2\} \Rightarrow \neg\,(x \leq 00) \equiv \neg$ pre $A1$. Thus, by law 5.8, we can replace A by $A1$ and B by $B1$ in $Op_{5.9}$, i.e. $A \vee B \sqsubseteq A1 \vee B1$.

Notice that although $A1$ and $B1$ specify different behaviours when $x = 0$, this just means that $A1 \vee B1$ is non-deterministic in this case. There is no inconsistency, since neither A nor B was defined for $x = 0$. ♠

We can relax the conditions further by allowing the preconditions of A and B (and thus of $A1$ and $B1$) to overlap, provided that $A1$ only adds behaviours for states that are not in the precondition of B, and $B1$ only adds behaviours for states that are not in the precondition of A. That is, any state in the preconditions of both $A1$ and B must also be in the precondition of A, and any state in the preconditions of both $B1$ and A must also be in the precondition of B. Since $A \sqsubseteq A1$, $B \sqsubseteq B1$, this means that any behaviour specified by $A1$ for initial states in the domain of B must be consistent with A, and, similarly, any behaviour specified by $B1$ for initial states in the domain of A must be consistent with B.

Law 5.9 If $A \sqsubseteq A1$, $B \sqsubseteq B1$, pre $A1 \wedge$ pre $B \Rightarrow$ pre A and pre $B1 \wedge$ pre $A \Rightarrow$ pre B, then $A \vee B \sqsubseteq A1 \vee B1$.

Proof

$$
\begin{array}{ll}
 & \text{pre } A \wedge B1 \\
\Rightarrow & \langle\ B1 \Rightarrow \text{pre } B1\ \rangle \\
 & \text{pre } A \wedge \text{pre } B1 \wedge B1 \\
\Rightarrow & \langle\ \text{pre } B1 \wedge \text{pre } A \Rightarrow \text{pre } B\ \rangle \\
 & \text{pre } B \wedge B1 \\
\Rightarrow & \langle\ B \sqsubseteq B1\ \rangle \\
 & B \\
\Rightarrow & \langle\ \text{logic}\ \rangle \\
 & A \vee B
\end{array}
$$

Similarly, we get pre $B \wedge A1 \Rightarrow A \vee B$, so by law 5.5, we have $A \vee B \sqsubseteq A1 \vee B1$. ♠

Example 5.10 Consider the operation $Op_{5.10} = A \vee B$, where:

$$
\begin{array}{lcl}
A & = & [x, x' : \mathbb{Z} \mid x \in \{0, -1\} \wedge x' = -1] \\
B & = & [x, x' : \mathbb{Z} \mid x \in \{0, 1\} \wedge x' = 1]
\end{array}
$$

We now refine A and B by weakening their preconditions but without adding behaviours for states only in the precondition of the other:

$$
\begin{array}{lcl}
A1 & = & [x, x' : \mathbb{Z} \mid x \leq 0 \wedge x' = -1] \\
B1 & = & [x, x' : \mathbb{Z} \mid x \geq 0 \wedge x' = 1]
\end{array}
$$

Now, we have:

$$\begin{array}{ll} & \text{pre } A1 \wedge \text{pre } B \\ \equiv & [x : \mathbb{Z} \mid x \leq 0 \wedge x \in \{0, 1\}] \\ \equiv & [x : \mathbb{Z} \mid x = 0] \\ \Rightarrow & [x : \mathbb{Z} \mid x \in \{0, -1\}] \\ \equiv & \text{pre } A \end{array} \qquad \begin{array}{ll} & \text{pre } B1 \wedge \text{pre } A \\ \equiv & [x : \mathbb{Z} \mid x \geq 0 \wedge x \in \{0, -1\}] \\ \equiv & [x : \mathbb{Z} \mid x \in \{0, 1\}] \\ \Rightarrow & [x : \mathbb{Z} \mid x = 0] \\ \equiv & \text{pre } B \end{array}$$

Thus, by law 5.9, we can replace A by $A1$ and B by $B1$ in $Op_{5.10}$, i.e. $A \vee B \sqsubseteq A1 \vee B1$.

In this case, although the preconditions of $A1$ and $B1$ overlap, they do so only for inputs in the preconditions of both A and B, so no conflict is introduced. ♠

We can relax the conditions even further, by just requiring that $A1$ does not add behaviours that conflict with B, and $B1$ does not add behaviours that conflict with A. That is, $A1$ can add behaviours for initial states that are in the precondition of B, but not of A, so long as those behaviours are ones permitted by B; similarly for $B1$.

Law 5.10 If $A \sqsubseteq A1$, $B \sqsubseteq B1$, $\text{pre } A1 \wedge \neg \text{ pre } A \wedge B \sqsubseteq A1$, and $\text{pre } B1 \wedge \neg \text{ pre } B \wedge A \sqsubseteq B1$, then $A \vee B \sqsubseteq A1 \vee B1$.

Proof First note that:

$$\begin{array}{ll} & \text{pre } A1 \wedge \neg \text{ pre } A \wedge B \sqsubseteq A1 \\ \equiv & \langle \text{ definition 4.1, logic } \rangle \\ & \text{pre } (\text{pre } A1 \wedge \neg \text{ pre } A \wedge B) \wedge A1 \Rightarrow \text{pre } A1 \wedge \neg \text{ pre } A \wedge B \\ \equiv & \langle \text{ laws 3.5 and 3.1, pre } A1 \text{ and } \neg \text{ pre contain no variables in } S' \rangle \\ & \text{pre } A1 \wedge \neg \text{ pre } A \wedge \text{pre } B \wedge A1 \Rightarrow \text{pre } A1 \wedge \neg \text{ pre } A \wedge B \\ \equiv & \langle \text{ logic, } A1 \Rightarrow \text{pre } A1 \rangle \\ & \neg \text{ pre } A \wedge \text{pre } B \wedge A1 \Rightarrow B \end{array}$$

Similarly, $\text{pre } B1 \wedge \neg \text{ pre } B \wedge A \sqsubseteq B1$ is equivalent to $\neg \text{ pre } B \wedge \text{pre } A \wedge B1 \Rightarrow A$.

Now, we have:

$$\begin{array}{ll} & \text{pre } A \wedge B1 \\ \equiv & \langle \text{ logic } \rangle \\ & (\text{pre } A \wedge \neg \text{ pre } B \wedge B1) \vee (\text{pre } A \wedge \text{pre } B \wedge B1) \\ \Rightarrow & \langle \text{ pre } A \wedge \neg \text{ pre } B \wedge B1 \Rightarrow A, B \sqsubseteq B1 \rangle \\ & A \vee B \end{array}$$

Similarly, we get $\text{pre } B \wedge A1 \Rightarrow A \vee B$, so by law 5.5, we have $A \vee B \sqsubseteq A1 \vee B1$. ♠

This law is more complex to apply than the previous ones, because it involves reasoning about the effects of the two operations not just their preconditions.

Example 5.11 Consider the operation $Op_{5.11} = A \vee B$, where:

$$A = [x, x' : \mathbb{Z} \mid -20 \le x \le 5 \wedge x \le x' \le 15]$$
$$B = [x, x' : \mathbb{Z} \mid -5 \le x \le 20 \wedge -15 \le x' \le x]$$

We can refine A and B so as to both decrease non-determinism and increase termination:

$$A1 = [x, x' : \mathbb{Z} \mid -30 \le x \le 10 \wedge x' = \textbf{if } x < -15 \textbf{ then } x + 1 \textbf{ else } x]$$
$$B1 = [x, x' : \mathbb{Z} \mid -10 \le x \le 30 \wedge x' = \textbf{if } x > 15 \textbf{ then } x - 1 \textbf{ else } x]$$

In this case, we have:

$$\begin{array}{ll}
 & \text{pre } A1 \wedge \neg \text{ pre } A \wedge B \\
\equiv & [x, x' : \mathbb{Z} \mid -30 \le x \le 10 \wedge \neg\, (-20 \le x \le 5) \wedge -10 \le x \le 30 \wedge x' = \textbf{if } x > 15 \textbf{ then } x - 1 \textbf{ else } x] \\
\equiv & [x, x' : \mathbb{Z} \mid 5 < x \le 10 \wedge x' = \textbf{if } x > 10 \textbf{ then } x - 1 \textbf{ else } x] \\
\equiv & [x, x' : \mathbb{Z} \mid 5 < x \le 10 \wedge x' = x] \\
\sqsubseteq & A1
\end{array}$$

and

$$\begin{array}{ll}
 & \text{pre } B1 \wedge \neg \text{ pre } B \wedge A \\
\equiv & [x, x' : \mathbb{Z} \mid -10 \le x \le 30 \wedge \neg\, (-5 \le x \le 20) \wedge -20 \le x \le 5 \wedge x' = \textbf{if } x < -15 \textbf{ then } x + 1 \textbf{ else } x] \\
\equiv & [x, x' : \mathbb{Z} \mid -10 \le x < -5 \wedge x' = \textbf{if } x < -10 \textbf{ then } x + 1 \textbf{ else } x] \\
\equiv & [x, x' : \mathbb{Z} \mid -10 \le x < -5 \wedge x' = x] \\
\sqsubseteq & B1
\end{array}$$

Thus, by law 5.10, we can replace A by $A1$ and B by $B1$ in $Op_{5.11}$, i.e. $A \vee B \sqsubseteq A1 \vee B1$.

In this example, refining A to $A1$ augments the precondition of A to include initial states that are in the precondition of B ($5 < x \le 10$) as well as ones that are not ($-20 < x < -15$). For initial states that are in the precondition of B, $A1$ leaves x unchanged, which is consistent with the behaviour specified by B. Likewise for $B1$. ♠

Finally, we note that refining the components of a disjunction is safe if both components are refined to the same thing.

Law 5.11 If $A \sqsubseteq C$ and $B \sqsubseteq C$, then $A \vee B \sqsubseteq C \vee C \equiv C$.

Proof

$$\begin{array}{ll}
 & \text{pre } A \wedge C \\
\equiv & \quad \langle\ A \sqsubseteq C\ \rangle \\
 & A \\
\Rightarrow & \quad \langle\ \text{logic}\ \rangle \\
 & A \vee B
\end{array}$$

Similarly, we get pre $B \wedge C \Rightarrow A \vee B$, so by law 5.5, we have $A \vee B \sqsubseteq C \vee C$, and $C \vee C \equiv C$ because $\vee$ is idempotent (law 4.1(i)). ♠

Example 5.12 Consider the operation $Op_{5.12} = A \vee B$, where:

$$A = [x, x' : \mathbb{Z} \mid x = 0 \wedge even(x')]$$
$$B = [x, x' : \mathbb{Z} \mid x = 1 \wedge odd(x')]$$

Now, suppose we refine both A and B by:

$$C = [x, x' : \mathbb{Z} \mid x' = x]$$

Since A and B are both refined by C, by law 5.11, we can replace both A and B by C in $Op_{5.12}$, i.e. $A \vee B \sqsubseteq C \vee C \equiv C$. ♠

The reason why disjunction is not monotonic is inherent in the fact that disjunction involves a non-deterministic choice between two partial operations, and increasing the domain of one operation can introduce conflicts with the other. There appears to be no way to avoid this problem without significantly altering the meaning of the disjunction operator (e.g. see [HR00]). One possibility, if we separate preconditions from postconditions, is to use a different operation, $\uplus$, defined so that, for $A = [\Delta S \mid PreA \mid PostA]$ and $B = [\Delta S \mid PreA \mid PostA]$, $A \uplus B \equiv [\Delta S \mid PreA \vee PreB \mid (PreA \Rightarrow PostA) \wedge (PreB \Rightarrow PostB)]$, which behaves like A for inputs in the precondition of A and like B for inputs in the precondition of B, and like $A \wedge B$ for inputs in the preconditions of both A and B. This operation is a join, and is monotonic and always exists in this semantics (cf. [AW94, Mor94a], where similar operations are defined within the refinement calculus).

6 Discussion

We have extended previous results concerning monotonicity of the main Z schema operators to explain just how these operators fail to be monotonic, and identified various cases where components can be safely replaced by their refinements. Although the schema calculus and refinement in Z have been discussed extensively in the literature, these kinds of results do not seem to have appeared before, with the exception of [CW98] where a law similar to the ones presented here is used to justify turning a Z composition into a (demonic) sequential composition. [Gro00] gives results similar to those in Section 5.2 are for a program disjunction operator in the refinement calculus.

These results raise the possibility of performing some refinement steps on a specification, without removing the schema operators and before moving to a more procedural notation, and might well be used in combination with the techniques described in [CW98]. Some of the results rely only on properties of individual components, or on the preconditions of components, and are thus easy to apply; others give stronger results but require reasoning about both components, which makes them more difficult to apply. It remains to be seen whether these results turn out to be useful in practice, and an obvious next step is to examine larger examples and case studies to see how often the special cases described in this paper occur.

We have focussed on conjunction and disjunction because they are the most often used schema operators and suffice to illustrate the kinds of properties we are interested in, and there are nice symmetries between them. Sequential composition displays some of the properties of both, being like conjunction in its first argument and like disjunction in its second argument, and leads to similar special cases. The other schema operators are typically either straightforwardly monotonic (e.g. precondition and renaming) or pathologically non-monotonic (e.g. negation and equivalence), so don't have such interesting properties. We intend to elaborate on these other operators in a later paper, and also to investigate how these results can be extended to promotion (cf. [Woo92, WD96]) and data refinement. The results for disjunction presented here all extend straightforwardly to data refinement, but those for conjunction do not (because existential quantification distributes over disjunction but not over conjunction), so it remains to be seen whether interesting cases can be found for conjunction.

We found it useful to introduce restricted forms of refinement that just modify the precondition or the postcondition, and identified properties of these relations that helped in proving some of the main results. These refinement relations may prove to be of interest in their own right, and are worthy of further investigation. Although we have conducted all of our proofs in terms of the proof obligations given in [Spi88], some of the proofs may be easier in the relational model given in [WD96], and it would be interesting to compare them.

Acknowledgements

This work was motivated by the quest of Martin Henson and Steve Reeves [HR00] to find monotonic versions of schema conjunction and disjunction. Thanks to Martin and Steve, and to Ana Cavalcanti, Eerke Boiten and Jim Woodcock, for many fruitful discussions and comments.

References

[AW94] Mike Ainsworth and Peter Wallis. Co-refinement. In David Till, editor, *6th Refinement Workshop*, pages 151–166. Springer-Verlag, 1994.

[BDBS99] Eerke A. Boiten, John Derrick, Howard Bowman, and Maarten Steen. Constructive consistency checking for partial specification in Z. *Science of Computer Programming*, 35(1):29–75, 1999.

[BEM92] Noureddine Boudriga, Fathi Elloumi, and Ali Mili. On the lattice of specifications: Applications to a specification methodology. *Formal Aspects of Computing*, 4(6):544–571, 1992.

[BvW98] Ralph-Johan Back and Joakim von Wright. *Refinement Calculus: A Systematic Introduction.* Graduate Texts in Computer Science. Springer-Verlag, 1998.

[CW98] A. L. C. Cavalcanti and J. C. P. Woodcock. ZRC – a refinement calculus for Z. *Formal Aspects of Computing*, 10(3):267–289, 1998.

[DB01] John Derrick and Eerke Boiten. *Refinement in Z and Object-Z: Foundations and Advanced Applications.* Springer-Verlag, 2001.

[DHR01] Moshe Deutsch, Martin C. Henson, and Steve Reeves. A proof-theoretic analysis of partial relation refinement I. Draft paper, 2001.

[Dil90] Antoni Diller. *Z: An Introduction to Formal Methods.* Wiley, 1990. Second edition, 1992.

[Gro00] Lindsay Groves. *Evolutionary Software Development in the Refinement Calculus.* PhD thesis, Victoria University of Wellington, 2000.

[HR00] Martin C. Henson and Steve Reeves. Program development and specification refinement in the schema calculus. In J.P. Bowen, S. Dunne, A. Galloway, and S. King, editors, *Proceedings of ZB2000: Formal Specification and Development in Z and B*, number 1878 in LNCS, pages 344–362. Springer, September 2000.

[Kin90] S. King. Z and the refinement calculus. In D. Bjørner, C. A. R. Hoare, and H. Langmaack, editors, *VDM and Z – Formal Methods in Software Development*, volume 428 of *Lecture Notes in Computer Science*, pages 164–188. VDM-Europe, Springer-Verlag, 1990.

[Mah99] Brendan Mahony. The least conjunctive refinement and promotion in the refinement calculus. *Formal Aspects of Computing*, 11:75–105, 1999.

[Mor94a] Carroll Morgan. The cuppest capjunctive capping, and Galois. In A. W. Roscoe, editor, *A Classical Mind: Essays in Honour of C. A. R. Hoare*, chapter 19, pages 317–332. Prentice Hall, 1994.

[Mor94b] Carroll Morgan. *Programming from Specifications*. Prentice Hall, second edition, 1994.

[PST91] Ben Potter, Jane Sinclair, and David Till. *An Introduction to Formal Specification and Z*. Prentice-Hall International, 1991. (Second edition, 1996).

[Spi88] Michael Spivey. *The Z Notation: A Reference Manual*. Prentice-Hall International, 1988. Second edition, 1992.

[War93] Nigel Ward. Adding specification constructors to the refinement calculus. In J. C. P. Woodcock and P. G. Larsen, editors, *Formal Methods Europe*, volume 670 of *Lecture Notes in Computer Science*, pages 652–670. Springer-Verlag, April 1993.

[WD96] Jim Woodcock and Jim Davies. *Using Z: Specification, Refinement and Proof*. Prentice-Hall International, 1996.

[Woo89] J. C. P. Woodcock. Calculating properties of Z specifications. *ACM SIGSOFT Software Engineering Notes*, 14(4):43–54, 1989.

[Woo92] J. C. P. Woodcock. Implementing promoted operations in Z. In Cliff B. Jones, Roger C. Shaw, and Tim Denvir, editors, *Proceedings of the 5th BCS Refinement Workshop*, pages 367–378. Springer-Verlag, 1992.

[Wor92] John Wordsworth. *Software Development with Z*. Addison-Wesley, 1992.