Refinement and the Z Schema Calculus

Lindsay Groves [1]

School of Mathematical and Computing Sciences
Victoria University of Wellington
Wellington, New Zealand

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 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 principal schema calculus operators are not monotonic with respect to refinement, 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. [9], [16], [7]); [21] is exceptional in discussing the interaction of refinement and promotion. Some authors suggest translating Z specifications into a procedural notation in a way that preserves some of the structure of the specifications (e.g. [12], [22], [20], [6]), 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

[1] Email: mailto:lindsay@mcs.vuw.ac.nz

like those of Z (e.g. [18], [13]),[2] or modified so that the schema operators are monotonic (e.g. [11]).

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 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 [3] 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 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 observations about the other schema operators and data refinement, and discussion of related and future work.

2 The Schema Calculus

We consider a subset of the schema calculus as defined in ZRM [17], 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 schemas are interpreted as defining operations 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: $[\Delta S \mid P]$.

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

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

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):

$$False_S = [\Delta S \mid false]$$

$$True_S = [\Delta S \mid true]$$

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 pre $A \equiv \exists S' \bullet A$ or pre $A \equiv [S \mid \exists S' \bullet P]$. Note that $A \Rightarrow$ 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 an operation $A = [\Delta S \mid P]$, we say:

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

(ii) A is *satisfiable* iff 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; i.e. that $A \wedge B$ terminates whenever A and B both terminate.

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.

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

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

3 Calculating Preconditions

This section presents some results about preconditions for operations defined by schema expressions that will be needed later. Throughout this section, we assume $A = [\Delta S \mid P]$ and $B = [\Delta S \mid Q]$.

First, we note that taking the precondition of an operation that doesn't constrain the post-state variables leaves the operation unchanged.

Law 3.1 pre $A \equiv A$ *if* $Vars(A) \cap S' = \varnothing$.

Proof.

$$
\begin{aligned}
& \text{pre } A \\
\equiv\ & [\Delta S \mid \exists S' \bullet P] \\
\equiv\ & [\Delta S \mid P] \\
\equiv\ & A
\end{aligned}
$$
□

In particular, law 3.1 shows that pre is idempotent; i.e. pre pre $A \equiv$ pre A.

3.1 Disjunction

Disjunction gives us a simple result ([19], [21, p210], [22, p125]).

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

Proof.

$$
\begin{aligned}
& \text{pre } (A \vee B) \\
\equiv\ & \text{pre } ([\Delta S \mid P] \vee [\Delta S \mid Q]) \\
\equiv\ & \text{pre } [\Delta S \mid P \vee Q] \\
\equiv\ & [\Delta S \mid \exists S' \bullet P \vee Q] \\
\equiv\ & [\Delta S \mid (\exists S' \bullet P) \vee (\exists S' \bullet Q)] \\
\equiv\ & [\Delta S \mid \exists S' \bullet P] \vee [\Delta S \mid \exists S' \bullet Q]) \\
\equiv\ & \text{pre } A \vee \text{pre } B
\end{aligned}
$$
□

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.

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

Law 3.3 $\text{pre}\ (A \wedge B) \Rightarrow \text{pre}\ A \wedge \text{pre}\ B$

Proof.

$$
\begin{aligned}
& \text{pre}\ (A \wedge B) \\
\equiv\ & \text{pre}\ ([\Delta S \mid P] \wedge [\Delta S \mid Q]) \\
\equiv\ & \text{pre}\ [\Delta S \mid P \wedge Q] \\
\equiv\ & [\Delta S \mid \exists S' \bullet P \wedge Q] \\
\Rightarrow\ & [\Delta S \mid (\exists S' \bullet P) \wedge (\exists S' \bullet Q)] \\
\equiv\ & [\Delta S \mid \exists S' \bullet P] \wedge [\Delta S \mid \exists S' \bullet Q] \\
\equiv\ & \text{pre}\ A \wedge \text{pre}\ B
\end{aligned}
$$

□

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

Law 3.4 $\text{pre}\ (A \wedge B) \equiv \text{pre}\ A \wedge \text{pre}\ B$, *if* $Vars(A) \cap Vars(B) = \varnothing$.

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.

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

Proof.

$$
\begin{aligned}
& \text{pre}\ (A \wedge B) \\
\equiv\ & [\Delta S \mid \exists S' \bullet P \wedge Q] \\
\equiv\ & [\Delta S \mid P \wedge (\exists S' \bullet Q)] \\
\equiv\ & A \wedge \text{pre}\ B
\end{aligned}
$$

□

In particular, law 3.5 shows that $\text{pre}\ (\text{pre}\ A \wedge B) \equiv \text{pre}\ A \wedge \text{pre}\ B$.

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 [17], and has been used by many other authors since (though some use turnstyle rather than implication). [21] and [7] instead give a semantic definition of refinement, based on a relational model ([8] shows that this definition is equivalent to Spivey's), and [6] give a definition in terms of a weakest precondition semantics. [5] The names "applicability" and "correctness" for the conditions are from [16]; Wordsworth [22] calls them "safety" and "liveness".

In this definition, inputs and outputs act in exactly the same way as pre- and post-state variables, respectively, 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)

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).

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) $\exists S \bullet \text{pre } A \wedge \neg \text{pre } B$
i.e. A is defined for some initial state for which B isn't, or

(ii) $\exists \Delta S \bullet \text{pre } A \wedge B \wedge \neg A$
i.e. for some initial state where A is defined, B admits some behaviour that A doesn't.

Note that the applicability condition limits the extent to which a postcondition can be strengthened. If a postcondition is strengthened so much that no possible outputs are left for some input, that input is no longer satisfies the precondition and the applicability condition fails. In particular, for any operation A whose precondition is not *False*, $A \not\sqsubseteq$ *False* because it fails the applicability condition; the correctness condition is satisfied.

We also note the following lattice properties of operation refinement:

Law 4.2

(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, and the meet of*

[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 4.1.

operations A and B is $\text{pre } A \wedge \text{pre } B \wedge (A \vee B)$. *(cf. [5], 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 [4].

4.1 Decomposing the refinement relation

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 4.3 (*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 $\forall \Delta S \bullet A \Rightarrow B$. This means that B can only modify A by increasing it precondition; i.e. by adding new behaviours for inputs not in the precondition of A.

Definition 4.4 (*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 $\forall S \bullet \text{pre } B \Rightarrow \text{pre } A$. This means that B can only modify A by reducing non-determinism; i.e. by removing some (but not all) behaviours for inputs in the precondition of A.

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

Law 4.5 $A \sqsubseteq_{pre} B$ *iff* $\forall \Delta S \bullet A \equiv \text{pre } A \wedge B$.

Proof.

$$
\begin{array}{ll}
 & A \sqsubseteq_{pre} B \\
\equiv & \langle \text{ definitions 4.3 and 4.1 } \rangle \\
 & (\forall S \bullet \text{pre } A \Rightarrow \text{pre } B) \wedge (\forall \Delta S \bullet \text{pre } A \wedge B \Rightarrow A) \wedge (\forall \Delta S \bullet A \Rightarrow B) \\
\equiv & \langle \text{ logic, } S' \text{ not free in pre } A \Rightarrow \text{pre } B \rangle \\
 & (\forall \Delta S \bullet (\text{pre } A \Rightarrow \text{pre } B) \wedge (\text{pre } A \wedge B \Rightarrow A) \wedge (A \Rightarrow B)) \\
\equiv & \langle \text{ logic, } A \Rightarrow \text{pre } A \rangle \\
 & \forall \Delta S \bullet A \equiv \text{pre } A \wedge B
\end{array}
$$

□

Law 4.6 $A \sqsubseteq_{post} B$ *iff* $\forall S \bullet \text{pre } A \equiv \text{pre } B$ *and* $\forall \Delta S \bullet B \Rightarrow A$.

Proof.

$$
\begin{aligned}
& A \sqsubseteq_{post} B \\
\equiv\ & \langle \text{ definitions 4.4 and 4.1 } \rangle \\
& (\forall S \bullet \text{pre } A \Rightarrow \text{pre } B) \wedge (\forall \Delta S \bullet \text{pre } A \wedge B \Rightarrow A) \wedge (\forall S \bullet \text{pre } B \Rightarrow \text{pre } A) \\
\equiv\ & \langle \text{ logic } \rangle \\
& (\forall S \bullet \text{pre } A \equiv \text{pre } B) \wedge (\forall \Delta S \bullet \text{pre } A \wedge B \Rightarrow A) \\
\equiv\ & \langle \text{ pre } A \equiv \text{pre } B,\ B \Rightarrow \text{pre } B \rangle \\
& (\forall S \bullet \text{pre } A \equiv \text{pre } B) \wedge (\forall \Delta S \bullet B \Rightarrow A)
\end{aligned}
$$

□

The following are some additional properties of $\sqsubseteq_{pre}$ and $\sqsubseteq_{post}$. We will not prove these here as they are not used in this paper.

- A pre-refinement of an operation A can be written as a disjunction of A and another operation C whose precondition is disjoint from that of A; i.e. $A \sqsubseteq_{pre} B$ iff B can be written as $A \vee C$, where pre $A \Rightarrow \neg$ pre C. Intuitively, C describes the behaviours added in the refinement and pre C contains just the new inputs added to the precondition.
- A post-refinement of an operation A can be written as a conjunction of A and another operation C whose precondition is contained in that of A and is consistent with A; i.e. $A \sqsubseteq_{post} B$ iff B can be written as $A \wedge C$, where pre $C \Rightarrow$ pre A, and A and C are consistent. Intuitively, C describes the additional constraint imposed on the outputs of A. The consistency condition ensures that this constraint is not so strong as to remove any inputs from the precondition of A.
- 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; i.e. $A \sqsubseteq_{pre} B$ and $A \sqsubseteq_{post} B$ iff $A \equiv B$.
- Any operation refinement can be obtained as a post-refinement followed by a pre-refinement; i.e. $A \sqsubseteq B$ iff $A \sqsubseteq_{post} E$ and $E \sqsubseteq_{pre} B$, for some E. Alternatively, $\sqsubseteq = \sqsubseteq_{post} \fatsemi \sqsubseteq_{pre}$.
- Any operation refinement can be obtained as a pre-refinement followed by a post-refinement; i.e. $A \sqsubseteq B$ iff $A \sqsubseteq_{pre} E$ and $E \sqsubseteq_{post} B$, for some E. Alternatively, $\sqsubseteq = \sqsubseteq_{pre} \fatsemi \sqsubseteq_{post}$.

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 a component 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 [3] 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

Schema conjunction is not monotonic with respect to refinement, because reducing non-determinism in its components can leave the operation with no choices for some input(s), thereby reducing the precondition of the operation and violating the applicability condition of definition 4.1.

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

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

and:

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

Thus, pre $(A \wedge B) \equiv x \in \mathbb{N}$ and pre $(A1 \wedge B1) \equiv \mathit{False}$, so pre $(A1 \wedge B1) \not\Rightarrow$ pre $(A1 \wedge B1)$.

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 $Op_{5.2}$.

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 of that component.

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

Law 5.3 *If* $A \sqsubseteq_{pre} A1$ *and* $B \sqsubseteq_{pre} B1$, *then* $A \wedge B \sqsubseteq_{pre} A1 \wedge B1$.

Proof. We prove that $A \wedge B \sqsubseteq_{pre} A1 \wedge B1$ using law 5.3.

$$
\begin{aligned}
& (A \sqsubseteq_{pre} A1) \wedge (B \sqsubseteq_{pre} B1) \\
\equiv\ & \quad \langle \text{ law 5.3 } \rangle \\
& (A \equiv \text{pre } A \wedge A1) \wedge (B \equiv \text{pre } B \wedge B1) \\
\Rightarrow\ & \quad \langle \text{ logic } \rangle \\
& A \wedge B \equiv \text{pre } A \wedge \text{pre } B \wedge A1 \wedge B1 \\
\equiv\ & \quad \langle \text{ law 5.3 } \rangle \\
& A \wedge B \sqsubseteq_{pre} A1 \wedge B1
\end{aligned}
$$

□

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

$$A = [x, x' : \mathbb{Z} \mid \mathit{even}(x) \wedge \mathit{even}(x')]$$

$$B = [x, x' : \mathbb{N}_1 \mid x \leq x' \leq 2 * x]$$

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 5.3, we can replace A by $A1$ and B by $B1$ in $Op_{5.4}$, i.e. $A \wedge B \sqsubseteq A1 \wedge B1$.

Law 5.3 shows that reducing non-determinism is the *only* way in which refining the components of a conjunction can lead to non-monotonicity, so we can guarantee monotonicity by ensuring that the refinements of the components do not reduce non-determinism. 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 for any input input the precondition on the conjunction. 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 $A \wedge B \sqsubseteq A1 \wedge B1$, 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 5.5 *$A \wedge B \sqsubseteq A1 \wedge B1$ if $A \sqsubseteq A1$, $B \sqsubseteq B1$, and $A1$ and $B1$ are consistent whenever A and B are.*

Proof. We prove that $A \wedge B \sqsubseteq A1 \wedge B1$ using definition 4.1.
Applicability: The assumption that $A1$ and $B1$ are consistent whenever A and B are consistent means that $\forall S \bullet \text{pre } (A \wedge B) \Rightarrow \text{pre } (A1 \wedge B1)$ holds.
Correctness:

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

Thus, by definition 4.1, we have $A \wedge B \sqsubseteq A1 \wedge B1$. □

Example 5.6 Consider the operation $Op_{5.6} = A \wedge B$, 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 5.5 we can replace A by $A1$ and B by $B1$ in $Op_{5.6}$, i.e. $A \wedge B \sqsubseteq A1 \wedge B1$.

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 5.7 *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 $Vars(P) \cap Var(Q) = \varnothing$, $Vars(P1) \subseteq Vars(P)$ 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 5.5 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 $A \wedge B \equiv [\Delta S;\ T \mid P \wedge Q] \equiv [\Delta S;\ T \mid P] \wedge [\Delta S;\ T \mid Q]$ and $A1 \wedge B1 \equiv [\Delta S;\ T \mid P1 \wedge Q1] \equiv [\Delta S;\ T \mid P1] \wedge [\Delta S;\ T \mid Q1]$. This is likely to be useful in practice, since conjunction is often used to combine operations on disjoint states to obtain an operation on a combined state.

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

$$A = [x, x' : \mathbb{N} \mid x < 100 \wedge x' > x]$$

$$B = [y, y' : \mathbb{N} \mid y < 10 \wedge y' \bmod 10 = y]$$

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 $Op_{5.8}$, 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, so overconstraining the outputs of an operation can reduce the domain of termination. 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 ([2,15]). 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 [11]. The results presented in this section show that non-monotonicity of conjunction is not a problem in certain circumstances, including the common case where the operations being conjoined operate on disjoint states.

5.2 Disjunction

Schema disjunction is not monotonic with respect to refinement, because weakening the precondition of one operation can add behaviours for initial states in the precondition of the other operation, thereby admitting previously excluded behaviours and violating the correctness condition of definition 4.1.

Example 5.9 Consider the operation $Op_{5.9} = A \vee B$, 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 $x = 0 \Rightarrow x \in \mathbb{N}$, and B is refined by:

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

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

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

and:

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

Thus, we have:

$$
\begin{aligned}
&\text{pre } (A \vee B) \wedge (A1 \vee B1) \\
&\equiv [x : \mathbb{N} \mid x = 0 \vee x = 1] \wedge [x, x' : \mathbb{N} \mid x' = 0] \vee [x, x' : \mathbb{N} \mid x' = 1] \\
&\equiv [x, x' : \mathbb{N} \mid (x = 0 \vee x = 1) \wedge (x' = 0 \vee x' = 1)] \\
&\equiv [x, x' : \mathbb{N} \mid x = 0 \wedge x' = 0 \vee x = 0 \wedge x' = 1 \vee x = 1 \wedge x' = 0 \vee x = 1 \wedge x' = 1]]
\end{aligned}
$$

So $A1 \vee B1$ admits the behaviours $x = 0 \wedge x' = 1$ and $x = 1 \wedge x' = 0$, which are 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 $Op_{5.9}$.

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 of that component.

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 5.10 *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 4.6. Given $A \sqsubseteq_{post} A1$ and $B \sqsubseteq_{post} B1$, we have:

(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 4.6.

(ii) $A1 \vee B1 \Rightarrow A \vee B$, since $A1 \Rightarrow A$ and $B1 \Rightarrow B$, by law 4.6.

Thus, by law 4.6, we have $A \wedge B \sqsubseteq_{post} A1 \wedge B1$. □

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

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

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

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

Since these refinements only modify the postconditions of A and B, by law 5.10, we can replace A by $A1$ and B by $B1$ in $Op_{5.11}$, i.e. $A \vee B \sqsubseteq A1 \vee B1$.

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. This can be specialised as shown in the following law, which says that the behaviour of $A \vee B$ must be consistent with $B1$ for inputs in the precondition of A (and of $B1$) and with $A1$ for inputs in the precondition of B (and of $A1$).

Law 5.12 *$A \vee B \sqsubseteq A1 \vee B1$ 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 5.12 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 5.13 *If $A \sqsubseteq A1$, $B \sqsubseteq B1$ and* pre $A1 \Rightarrow \neg$ pre $B1$, *then $A \vee B \sqsubseteq A1 \vee B1$.*

Proof. We prove that $A \vee B \sqsubseteq A1 \vee B1$ using law 5.12, so we need to show that pre $A \wedge B1 \Rightarrow A \vee B$ and pre $B \wedge A1 \Rightarrow A \vee B$.

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

Similarly, we get pre $B \wedge A1 \Rightarrow A \vee B$, so by law 5.12, we have $A \vee B \sqsubseteq A1 \vee B1$. □

Example 5.14 Consider the operation $Op_{5.14} = A \vee B$, 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 pre $A1 \equiv x < 0 \Rightarrow \neg\ (x > 0) \equiv \neg$ pre $B1$. Thus, by law 5.13, we can replace A by $A1$ and B by $B1$ in $Op_{5.14}$, i.e. $A \vee B \sqsubseteq A1 \vee B1$.

A special case of law 5.13 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 5.15 *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 $A \vee B \sqsubseteq A1 \vee B1$.*

Proof. With these assumptions, the conditions of law 5.13 are satisfied. □

We can apply law 5.15 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 5.16 Consider the operation $Op_{5.16} = A \lor B$, where:

$$A = [x, x' : \mathbb{Z} \mid x < x' \leq hi]$$

$$B = [y, y' : \mathbb{Z} \mid y > y' \geq lo]$$

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

$$A1 = [x, x' : \mathbb{Z} \mid x' = x + 1]$$

$$B1 = [x, x' : \mathbb{Z} \mid y' = y - 1]$$

Since A and B, and $A1$ and $B1$, operate on disjoint states, by law 5.15, we can replace A by $A1$ and B by $B1$ in $Op_{5.16}$, i.e. $A \lor B \sqsubseteq A1 \lor B1$.

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 allow the preconditions of the refinements to overlap outside of the preconditions of original components.

Law 5.17 *If* $A \sqsubseteq A1$, $B \sqsubseteq B1$, pre $A \Rightarrow \neg$ pre $B1$ *and* pre $B \Rightarrow \neg$ pre $A1$, *then* $A \lor B \sqsubseteq A1 \lor B1$.

Proof. We prove that $A \lor B \sqsubseteq A1 \lor B1$ using law 5.12, so we need to show that pre $A \land B1 \Rightarrow A \lor B$ and pre $B \land A1 \Rightarrow A \lor B$.

pre $A \land B1$

$\Rightarrow$ $\langle$ pre $A \Rightarrow \neg$ pre $B1$ and $B1 \Rightarrow$ pre $B1$ $\rangle$

False

$\Rightarrow$ $\langle$ logic $\rangle$

$A \lor B$

Similarly, we get pre $B \land A1 \Rightarrow A \lor B$, so by law 5.12, we have $A \lor B \sqsubseteq A1 \lor B1$. □

Example 5.18 Consider the operation $Op_{5.18} = A \lor B$, where:

$$A = [x, x' : \mathbb{Z} \mid x \in \{-1, -2\} \land x' = -1]$$

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

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

$$A1 = [x, x' : \mathbb{Z} \mid x \leq 0 \land x' = -1]$$

$$B1 = [x, x' : \mathbb{Z} \mid x \geq 0 \land x' = 1]$$

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 5.17, we can replace A by $A1$ and B by $B1$ in $Op_{5.18}$, i.e. $A \lor B \sqsubseteq A1 \lor B1$.

Notice that although $A1$ and $B1$ specify different behaviours when $x = 0$, this just means that $A1 \lor B1$ 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 5.19 *If* $A \sqsubseteq A1$, $B \sqsubseteq B1$, pre $A1 \land$ pre $B \Rightarrow$ pre A *and* pre $B1 \land$ pre $A \Rightarrow$ pre B, *then* $A \lor B \sqsubseteq A1 \lor B1$.

Proof. We prove that $A \lor B \sqsubseteq A1 \lor B1$ using law 5.12, so we need to show that pre $A \land B1 \Rightarrow A \lor B$ and pre $B \land A1 \Rightarrow A \lor B$.

pre $A \land B1$

$\Rightarrow$ $\langle\ B1 \Rightarrow$ pre $B1\ \rangle$

pre $A \land$ pre $B1 \land B1$

$\Rightarrow$ $\langle\ $ pre $B1 \land$ pre $A \Rightarrow$ pre $B\ \rangle$

pre $B \land B1$

$\Rightarrow$ $\langle\ B \sqsubseteq B1$, logic $\rangle$

$A \lor B$

Similarly, we get pre $B \land A1 \Rightarrow A \lor B$, so by law 5.12, we have $A \lor B \sqsubseteq A1 \lor B1$. □

Example 5.20 Consider the operation $Op_{5.20} = A \lor B$, where:

$$A = [x, x' : \mathbb{Z} \mid x \in \{0, -1\} \land x' = -1]$$

$$B = [x, x' : \mathbb{Z} \mid x \in \{0, 1\} \land x' = 1]$$

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

$$A1 = [x, x' : \mathbb{Z} \mid x \leq 0 \land x' = -1]$$
$$B1 = [x, x' : \mathbb{Z} \mid x \geq 0 \land x' = 1]$$

Now, we have:

pre $A1 \land$ pre B	pre $B1 \land$ pre A
$\equiv [x : \mathbb{Z} \mid x \leq 0 \land x \in \{0, 1\}]$	$\equiv [x : \mathbb{Z} \mid x \geq 0 \land x \in \{0, -1\}]$
$\equiv [x : \mathbb{Z} \mid x = 0]$	$\equiv [x : \mathbb{Z} \mid x \in \{0, 1\}]$
$\Rightarrow [x : \mathbb{Z} \mid x \in \{0, -1\}]$	$\Rightarrow [x : \mathbb{Z} \mid x = 0]$
$\equiv$ pre A	$\equiv$ pre B

Thus, by law 5.19, we can replace A by $A1$ and B by $B1$ in $Op_{5.20}$, i.e. $A \lor B \sqsubseteq A1 \lor B1$.

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 5.21 *If* $A \sqsubseteq A1$, $B \sqsubseteq B1$, pre $A1 \land \neg$ pre $A \land B \sqsubseteq A1$, *and* pre $B1 \land \neg$ pre $B \land A \sqsubseteq B1$, *then* $A \lor B \sqsubseteq A1 \lor B1$.

Proof. We prove that $A \lor B \sqsubseteq A1 \lor B1$ using law 5.12, so we need to show that pre $A \land B1 \Rightarrow A \lor B$ and pre $B \land A1 \Rightarrow A \lor B$.

First, note that:

pre $A1 \land \neg$ pre $A \land B \sqsubseteq A1$

$\equiv$ $\langle$ definition 4.1, logic $\rangle$

pre (pre $A1 \land \neg$ pre $A \land B$) $\land A1 \Rightarrow$ pre $A1 \land \neg$ pre $A \land B$

$\equiv$ $\langle$ laws 3.5 and 3.1, pre $A1$ and $\neg$ pre contain no variables in S' $\rangle$

pre $A1 \land \neg$ pre $A \land$ pre $B \land A1 \Rightarrow$ pre $A1 \land \neg$ pre $A \land B$

$\equiv$ $\langle$ logic, $A1 \Rightarrow$ pre $A1$ $\rangle$

$\neg$ pre $A \land$ pre $B \land A1 \Rightarrow B$

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

Now, we have:

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

Similarly, we get pre $B \wedge A1 \Rightarrow A \vee B$, so by law 5.12, 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 5.22 Consider the operation $Op_{5.22} = A \vee B$, where:

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

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

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

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

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

In this case, we have:

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

and:

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

Thus, by law 5.21, we can replace A by $A1$ and B by $B1$ in $Op_{5.22}$, 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 ($5 < x \leq 10$) as well as ones that are not ($-20 < x < -15$). For initial statesthat 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 operation.

Law 5.23 *If $A \sqsubseteq C$ and $B \sqsubseteq C$, then $A \vee B \sqsubseteq C \vee C \equiv C$.*

Proof. We prove that $A \vee B \sqsubseteq A1 \vee B1$ using law 5.12, so we need to show that pre $A \wedge B1 \Rightarrow A \vee B$ and pre $B \wedge A1 \Rightarrow A \vee B$.

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

Similarly, we get pre $B \wedge C \Rightarrow A \vee B$, so by law 5.12, we have $A \vee B \sqsubseteq C \vee C$, and $C \vee C \equiv C$ because $\vee$ is idempotent (law 4.2(i)). □

Example 5.24 Consider the operation $Op_{5.24} = A \vee B$, where:

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

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 5.23, we can replace both A and B by C in $Op_{5.24}$, 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 behaviours that conflict 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 [11]). 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. [1,14], where similar operations are defined within the refinement calculus). The results presented in this section show that non-monotonicity of disjunction is not a problem in various circumstances where no conflict is introduced. It seems likely that such conflict would be rare in practice.

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 identify 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 [6] where a law similar to the ones presented here is used to justify turning a Z composition into a (demonic) sequential composition. [10] gives results similar to those in Section 5.2 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 [6]. 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. [20,21]) 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 [17], some of the proofs may be easier in the relational model given in [21], and it would be interesting to compare them.

Acknowledgements

This work was motivated by the quest of Martin Henson and Steve Reeves [11] 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

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

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

[3] Rosalind Barden, Susan Stepney, and David Cooper. The use of Z. In J. E. Nicholls, editor, *Proc. 6th Z User Meeting, York 1991*, Workshops in Computing, pages 99–124. Springer-Verlag, 1992.

[4] 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.

[5] 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.

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

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

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

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

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

[11] 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.

[12] 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.

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

[14] 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.

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

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

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

[18] 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.

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

[20] 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.

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

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