Exploring Summation and Product Operators in the Refinement Calculus

R.J.R. Back and M.J. Butler
Dept. of Computer Science, Åbo Akademi
Turku, Finland

9 September 1994

Abstract

Product and summation operators for predicate transformers were introduced by Naumann [21] and by Martin [15] using category theoretic considerations. In this paper, we formalise these operators in the higher order logic approach to the refinement calculus of [4], and examine various algebraic properties of these operators. There are several motivating factors for this analysis. The product operator provides a model of simultaneous execution of statements, while the summation operator provides a simple model of late binding. We also generalise the product operator slightly to form an operator that corresponds to conjunction of specifications. We examine several applications of the these operators showing, for example, how a combination of the product and summation operators could be used to model inheritance in an object-oriented programming language.

1 Introduction

Dijkstra introduced weakest-precondition predicate transformers as a means of verifying total correctness properties of sequential programs [8]. In the *refinement calculus* of Back and others, specifications and programs are regarded uniformly as predicate transformers, and refinement laws are derived from properties of predicate transformers [1, 16, 19].

The refinement calculus provides various choice and assignment operators that are generalisations of Dijkstra's operators, and the applications of these operators are well-known. However, the applications of an operator representing simultaneous execution of program statements are less well developed in the refinement calculus. Such an operator was introduced by Naumann [21] and by Martin [15] using category theoretic considerations. This *product* operator combines predicate transformers by forming the cartesian product of their state spaces. In this paper, we examine the product operator using the higher-order logic formalisation of the refinement calculus of Back & von Wright [4]. We examine various distributivity and refinement preserving properties of the operator and show that it can be used to model simultaneous execution and to extend the state spaces of statements so they can be more easily matched with other statements. We also generalise the definition of the product operator slightly to form what we call a *fusion* operator and show that the product operator is a special case of the fusion operator. The fusion operator can also be applied to conjoining or amalgamating specification statements.

The *summation* (or *co-product*[1]) operator, which is the categorical dual of the product operator and combines statements by forming the disjoint union of their state spaces, is also described in [21] and [15]. The summation operator is a form of choice operator and we show that it is a special case of the existing choice operators of the refinement calculus. We show that this operator provides a simple yet powerful model of late binding, and that when combined with the product operator, provides a model of inheritance in an object-oriented programming language.

The paper is organised as follows. First the refinement calculus basics are described. Then Section 3 examines properties of the summation operator, while Section 4 examines properties of the product and fusion operators. In Section 5 we look briefly at the categorical properties of the summation and product operators. Section 6 looks at various applications of the operators including simultaneous execution, extending state spaces, late binding, and inheritance.

[1] The direction of arrows in [21] and [15] are the reverse of what we use, so our product operator corresponds to their co-product operator, and vice versa.

2 Refinement Calculus Basics

A *predicate* over a set of states Σ is a boolean function $p : \Sigma \to \mathbf{Bool}$ which assigns a truth value to each state. The set of predicates on Σ is denoted $\mathcal{P}\,\Sigma$:

$$\mathcal{P}\,\Sigma \quad \widehat{=} \quad \Sigma \to \mathbf{Bool}.$$

We define the *entailment ordering* on predicates by pointwise extension: for $p, q : \mathcal{P}\,\Sigma$,

$$p \leq q \quad \widehat{=} \quad (\forall \sigma : \Sigma \cdot p\ \sigma \ \Rightarrow\ q\ \sigma).$$

The identically false predicate is denoted $\bot$, and the identically true predicate is denoted $\top$. Negation, conjunction, and disjunction of (similarly-typed) predicates are defined by pointwise extension, so that, e.g.,

$$(p \wedge q)\ \sigma \quad \widehat{=} \quad (p\ \sigma\ \wedge\ q\ \sigma).$$

Predicates form a complete lattice under the entailment ordering. Conjunction and disjunction represent meet and join respectively, while $\top$ and $\bot$ represent top and bottom respectively.

A *relation* from Σ to Γ is a function $P : \Sigma \to \mathcal{P}\,\Gamma$ that maps each state σ to a predicate on Γ. We write

$$\Sigma \leftrightarrow \Gamma \quad \widehat{=} \quad \Sigma \to \mathcal{P}\,\Gamma.$$

This view of relations is isomorphic to viewing them as predicates on the cartesian space $\Sigma \times \Gamma$. The domain and range of relation P, are denoted $dom\,P$ and $ran\,P$ respectively. Conjunction and disjunction of relations is defined pointwise, so that, e.g., $(P \wedge Q)\ \sigma\ \gamma \ = \ (P\ \sigma\ \gamma) \wedge (Q\ \sigma\ \gamma)$. We write $P; Q$ for the relational composition of $P : \Sigma \leftrightarrow \Gamma$ and $Q : \Gamma \leftrightarrow \Lambda$, and P^{-1} for the inverse relation. For function $f : \Sigma \to \Gamma$, we write f^{-1} for the relation $(\lambda\gamma \cdot (\lambda\sigma \cdot \gamma = f\sigma)) : \Gamma \leftrightarrow \Sigma$. A partial function $g : \Sigma \to_p \Gamma$ is a relation from Σ to Γ such that $g\ \sigma\ \gamma\ \wedge\ g\ \sigma\ \gamma' \ \Rightarrow\ \gamma = \gamma'$.

A *predicate transformer* is a function $S : \mathcal{P}\,\Gamma \to \mathcal{P}\,\Sigma$ from predicates to predicates. We write,

$$\Sigma \longmapsto \Gamma \quad \widehat{=} \quad \mathcal{P}\,\Gamma \ \to\ \mathcal{P}\,\Sigma.$$

Note the reversal of Γ and Σ: program statements in the refinement calculus are identified with weakest-precondition predicate transformers that map a postcondition $q : \mathcal{P}\,\Gamma$ to the weakest precondition $p : \mathcal{P}\,\Sigma$ such that the program is guaranteed to terminate in a final state satisfying q whenever the initial state satisfies p. For program statement $S : \Sigma \longmapsto \Gamma$, we say that S has *source* Σ and *target* Γ. Programs need not have identical initial and final state spaces, though if they do, we write $S : \Delta(\Sigma)$ instead of $S : \Sigma \longmapsto \Sigma$. Note that, in order to aid intuition, we sometimes discuss commands from an operational viewpoint, though all our formal reasoning is always in terms of the predicate transformer semantics.

The *refinement ordering* on predicates is defined by pointwise extension: for $S, T : \Sigma \longmapsto \Gamma$,

$$S \leq T \quad \widehat{=} \quad (\forall q : \mathcal{P}\,\Gamma \cdot S\ q\ \leq\ T\ q).$$

The refinement ordering on predicates models the notion of total-correctness preserving program refinement. A total-correctness specification is typically given as a precondition-postcondition pair $(pre, post)$, and program (i.e., predicate transformer) S satisfies $(pre, post)$ if $pre \leq (S\ post)$. Now, for programs S and T, $S \leq T$ holds if and only if T satisfies any specification satisfied by S.

Predicate transformers form a complete lattice under the refinement ordering. The bottom element is the predicate transformer *abort* that maps each postcondition to $\bot$, and the top element is the predicate transformer *magic* that maps each postcondition to $\top$. The *abort* statement is never guaranteed to terminate, while the *magic* statement is *miraculous* since it is always guaranteed to establish any postcondition. A miraculous statement cannot be implemented. For statement S, $halt\ S \mathrel{\widehat{=}} S\ \top$ describes those initial states under which S is guaranteed to terminate, while $gd\ S \mathrel{\widehat{=}} \neg\,(S\ \bot)$ describes those initial states under which S behaves non-miraculously.

Conjunction and disjunction of (similarly-typed) predicate transformers is defined pointwise, so that, e.g., $(S \wedge T)\ q = (S\ q) \wedge (T\ q)$. Conjunction of statements models *demonic nondeterministic choice* between executing S and T (i.e., each alternative must establish the postcondition), whereas disjunction models *angelic nondeterministic choice* (i.e., some alternative must establish the postcondition). If $halt\ S =$

$\neg$ *halt* T, then $S \vee T$ becomes a deterministic choice, while if $gd\ S = \neg\ gd\ T$, then $S \wedge T$ becomes a deterministic choice.

Sequential composition of program statements is modelled by functional composition of predicate transformers, i.e., for $S : \Sigma \longmapsto \Gamma,\ \ T : \Gamma \longmapsto \Lambda,\ \ p : \mathcal{P}\,\Lambda$,

$$S;T\ p \quad \widehat{=} \quad S\ (T\ p).$$

The program statement $skip_\Sigma$ is modelled by the identity predicate transformer on $\mathcal{P}\,\Sigma$.

Given a relation $P : \Sigma \leftrightarrow \Gamma$, the *angelic update statement* $\{P\} : \Sigma \longmapsto \Gamma$ and *demonic update statement* $[P] : \Sigma \longmapsto \Gamma$ are defined by

$$\begin{aligned} \{P\}\ q\ \sigma \quad &\widehat{=} \quad (\exists \gamma : \Gamma \cdot (P\ \sigma\ \gamma) \wedge (q\ \gamma)) \\ [P]\ q\ \sigma \quad &\widehat{=} \quad (\forall \gamma : \Gamma \cdot (P\ \sigma\ \gamma) \Rightarrow (q\ \gamma)). \end{aligned}$$

When started in a state σ, $\{P\}$ angelically chooses a new state γ such that $P\sigma\gamma$ holds, while $[P]$ demonically chooses a new state γ such that $P\sigma\gamma$ holds. If no such new state exists then $\{P\}$ aborts, while $[P]$ behaves as *magic*.

For predicate $p : \mathcal{P}\,\Sigma$, let $\overline{p} : \Sigma \leftrightarrow \Sigma$ be the corresponding test relation for p, i.e., $(\lambda\sigma \cdot \lambda\sigma' \cdot \sigma = \sigma' \wedge p\ \sigma)$. Then we write $\{p\}$ for $\{\overline{p}\}$ and $[p]$ for $[\overline{p}]$. $\{p\}$ models the *assert* statement that behaves as *skip* if p holds, otherwise it aborts. $[p]$ models the *guard* statement that behaves as *skip* if p holds, otherwise it behaves as *magic*.

Given a function $f : \Sigma \rightarrow \Gamma$, the *deterministic update statement* $\langle f \rangle : \Sigma \longmapsto \Gamma$ is defined by

$$\langle f \rangle\ q\ \sigma \quad \widehat{=} \quad q\ (f\ \sigma).$$

Let $\tilde{f}$ be the the deterministic relation corresponding to function f, i.e., $(\lambda\sigma \cdot \lambda\gamma \cdot f\ \sigma = \gamma)$. Then we have

$$\{\tilde{f}\} \ = \ \langle f \rangle \ = \ [\tilde{f}].$$

Ordinary program constructs such conditionals, recursions, and assignments may be modelled using the basic operators presented above. For example, **if** g **then** S **else** T **fi** may be modelled by $[g];S\ \wedge\ [\neg\ g];T$, or alternatively by, $\{g\};S\ \vee\ \{\neg\ g\};T$. Program variables may be modelled as cartesian components of the state space, and a multiple assignment $x, y := e, f$ in a state space with three components, representing program variables (x, y, z), may be modelled by the deterministic update:

$$\langle \lambda x, y, z \cdot (e, f, z) \rangle.$$

It is easy to show, for predicates *pre* and *post*, that $\{pre\};[post]$ is the least statement satisfying the specification pair $(pre, post)$. Thus, implementing the specification $(pre, post)$ involves constructing a statement S such that $\{pre\};[post] \leq S$. If *post* is a relation rather than a predicate, we can model postconditions on the before and after states in the manner of VDM [14] and Z [24]. Rules for the stepwise refinement of specification statements of the form $\{pre\};[post]$ into more familiar program constructs may be found in [1, 16, 19].

All predicate transformers S constructed using the operators described above will be *monotonic*, i.e., $p \leq q\ \Rightarrow\ S\ p \leq S\ q$. A predicate transformer S is *bottom homomorphic* if $S\ \bot = \bot$, and *top homomorphic* if $S\ \top = \top$. *Conjunctive* predicate transformers form a subset of the monotonic predicate transformers satisfying $S\ (\forall i : I \cdot q_i) = (\forall i : I \cdot S\ q_i)$, for non-empty I. S is *universally conjunctive* if it is conjunctive and top homomorphic. *Disjunctive* and *universally disjunctive* predicate transformers are defined dually. A predicate transformer is *continuous* if it is disjunctive over non-empty chains of predicates. Demonic update statements are universally conjunctive, angelic update statements are universally disjunctive, and deterministic update statements are both. The operators $\wedge$ and ; preserve the conjunctivity of their operands, while $\vee$ and ; preserve the disjunctivity of their operands. Also, each of $\wedge$, $\vee$, and ; preserve refinement of monotonic predicate transformers, e.g., $S \leq S'\ \Rightarrow\ S;T \leq S';T$.

In Dijkstra's original calculus, all statements were conjunctive and non-miraculous. Also, for conjunctive statements, disjunctivity corresponds to determinism, while continuity corresponds to bounded nondeterminism [8].

3 Summation Operators

In this section, we describe two summation operators for predicate transformers: the first only sums the operand sources, while the second sums both the operand sources and the operand targets.

The sum (or disjoint union) of two state spaces Σ_1 and Σ_2 is written $\Sigma_1 + \Sigma_2$. Associated with the summation are two *injections* ϕ_1 and ϕ_2 which map elements of the base type to elements of the summation ($A \rightarrow_\iota B$ represents the set of injective functions from A to B):

$$\phi_1 : \Sigma_1 \rightarrow_\iota \Sigma_1 + \Sigma_2 \tag{1}$$
$$\phi_2 : \Sigma_2 \rightarrow_\iota \Sigma_1 + \Sigma_2. \tag{2}$$

Since any element of $\Sigma_1 + \Sigma_2$ must either come from Σ_1 or Σ_2, but not both, the injections also satisfy:

$$(ran\, \phi_1, ran\, \phi_2) \text{ partition } \Sigma_1 + \Sigma_2. \tag{3}$$

We shall assume that all injection pairs (ϕ_1, ϕ_2) associated with summations satisfy (1), (2), and (3).

Since ϕ_1 and ϕ_2 are injective, their inverses ϕ_1^{-1} and ϕ_2^{-1} are partial functions:

$$\phi_1^{-1} : \Sigma_1 + \Sigma_2 \rightarrow_p \Sigma_1 \qquad \phi_2^{-1} : \Sigma_1 + \Sigma_2 \rightarrow_p \Sigma_2.$$

For $\sigma \in \Sigma_1 + \Sigma_2$, $(\phi_1^{-1}\sigma)$ is only defined if $\sigma \in (dom\, \phi_1^{-1})$ and similarly for $(\phi_2^{-1}\sigma)$.

Summation of predicates is given by:

Definition 1 *For $p_1 : \mathcal{P}\,\Sigma_1,\ p_2 : \mathcal{P}\,\Sigma_2,\ \sigma : \Sigma_1 + \Sigma_2$,*

$$\begin{array}{rcl} (p_1 + p_2)\ \sigma & \mathrel{\widehat{=}} & \sigma \in (dom\, \phi_1^{-1}) \wedge p_1(\phi_1^{-1}\sigma)\ \vee \\ & & \sigma \in (dom\, \phi_2^{-1}) \wedge p_2(\phi_2^{-1}\sigma). \end{array}$$

Given an injection $\phi_1 : \Sigma_1 \rightarrow_\iota \Sigma_1 + \Sigma_2$, the deterministic update

$$\langle \phi_1 \rangle : \Sigma_1 \longmapsto \Sigma_1 + \Sigma_2 \qquad (\ =\ \mathcal{P}\,(\Sigma_1 + \Sigma_2) \rightarrow \mathcal{P}\,\Sigma_1)$$

maps predicates in the summation type to predicates in the first base type. The updates can be used to select the components of a summation of predicates:

Theorem 1 *For $p_1 : \mathcal{P}\,\Sigma_1,\ p_2 : \mathcal{P}\,\Sigma_2$,*

$$\langle \phi_1 \rangle\ (p_1 + p_2)\ =\ p_1 \qquad \text{and} \qquad \langle \phi_2 \rangle\ (p_1 + p_2)\ =\ p_2.$$

Proof:

$$\begin{array}{rcl} \langle \phi_1 \rangle\ (p_1 + p_2)\ \sigma_1 & = & (p_1 + p_2)\ (\phi_1\ \sigma_1) \\ & = & \qquad \{\text{Definition 1 and partition property (3)}\} \\ & & p_1\ \sigma_1. \end{array}$$

□

The following theorem shows that a predicate in the summation type can always be separated into a summation of predicates:

Theorem 2 *For $p : \mathcal{P}\,(\Sigma_1 + \Sigma_2)$,*

$$p\ =\ \langle \phi_1 \rangle\ p\ +\ \langle \phi_2 \rangle\ p.$$

Proof: for $\sigma : \Sigma_1 + \Sigma_2$,

$$\begin{array}{cl} & (\langle \phi_1 \rangle\ p\ +\ \langle \phi_2 \rangle\ p)\ \sigma \\ = & \sigma \in (dom\, \phi_1^{-1})\ \wedge\ (\langle \phi_1 \rangle\ p)\ (\phi_1^{-1}\sigma)\ \ \vee\ \ \sigma \in (dom\, \phi_2^{-1})\ \wedge\ (\langle \phi_2 \rangle\ p)\ (\phi_2^{-1}\sigma) \\ = & \sigma \in (dom\, \phi_1^{-1})\ \wedge\ p\ (\phi_1\ (\phi_1^{-1}\sigma))\ \ \vee\ \ \sigma \in (dom\, \phi_2^{-1})\ \wedge\ p\ (\phi_2\ (\phi_2^{-1}\sigma)) \\ = & \sigma \in (dom\, \phi_1^{-1})\ \wedge\ p\ \sigma\ \ \vee\ \ \sigma \in (dom\, \phi_2^{-1})\ \wedge\ p\ \sigma \\ = & \qquad \{\text{Partition property of injection pairs}\} \\ & p\ \sigma. \end{array}$$

□

3.1 Summation of Predicate Transformers

The summation of predicate transformers S_1 and S_2 is written $S_1 \oplus S_2$, where S_1 and S_2 have the same target, but possibly distinct sources:

DEFINITION **2** *For* $S_1 : \Sigma_1 \longmapsto \Gamma$, $\quad S_2 : \Sigma_2 \longmapsto \Gamma$, $S_1 \oplus S_2$ *is of type* $\Sigma_1 + \Sigma_2 \longmapsto \Gamma$, *where for* $q : \mathcal{P}\,\Gamma$*:*

$$(S_1 \oplus S_2)\ q \quad \widehat{=} \quad (S_1\ q)\ +\ (S_2\ q).$$

The selection and separation properties of the summation operator follow directly from Theorems 1 and 2:

THEOREM **3** *For* $S_1 : \Sigma_1 \longmapsto \Gamma,\ S_2 : \Sigma_2 \longmapsto \Gamma,\ S : \Sigma_1 + \Sigma_2 \longmapsto \Gamma$,

$$\langle \phi_1 \rangle; (S_1 \oplus S_2) \ = \ S_1 \quad \textit{and} \quad \langle \phi_2 \rangle; (S_1 \oplus S_2) \ = \ S_2$$
$$S \ = \ \langle \phi_1 \rangle; S \ \oplus \ \langle \phi_2 \rangle; S.$$

Summation of predicate transformers is a form of choice operator since the effect of executing $S_1 \oplus S_2$ in some initial state σ depends on the base type of σ: if it is of type Σ_1, then S_1 is executed, while if it is of type Σ_2, then S_2 is executed. This may be seen more clearly in the following theorem where the summation operator is characterised purely in terms of the existing choice operators. The theorem uses the following angelic and demonic updates formed from the inverse injections:

$$\{\phi_1^{-1}\} : \Sigma_1 + \Sigma_2 \longmapsto \Sigma_1 \qquad\qquad [\phi_1^{-1}] : \Sigma_1 + \Sigma_2 \longmapsto \Sigma_1.$$

THEOREM **4** *For* $S_1 : \Sigma_1 \longmapsto \Gamma,\ S_2 : \Sigma_2 \longmapsto \Gamma$,

$$\begin{aligned} S_1 \oplus S_2 \ &= \ \{\phi_1^{-1}\}; S_1 \ \vee \ \{\phi_2^{-1}\}; S_2 \\ &= \ [\phi_1^{-1}]; S_1 \ \wedge \ [\phi_2^{-1}]; S_2. \end{aligned}$$

It is also clear from Theorem 4 that the choice between S_1 and S_2 is deterministic since ϕ_1^{-1} and ϕ_2^{-1} have disjoint domains, so that

$$\textit{halt}\ \{\phi_1^{-1}\} = \neg\ \textit{halt}\ \{\phi_2^{-1}\} \quad \text{and} \quad \textit{gd}\ [\phi_1^{-1}] = \neg\ \textit{gd}\ [\phi_2^{-1}].$$

Before proving Theorem 4 we note the following, easily-proven lemmas:

LEMMA **1** *For injective function* $f : \Sigma \rightarrow_\iota \Gamma$,

$$\langle f \rangle; \{f^{-1}\} \ = \ \langle f \rangle; [f^{-1}] \ = \ \textit{skip}.$$

LEMMA **2** *For function* $f : \Sigma \rightarrow \Gamma$, *relation* $P : \Gamma \leftrightarrow \Lambda$, *such that* $\textit{ran}\ f \cap \textit{dom}\ P = \emptyset$,

$$\langle f \rangle; \{P\} \ = \ \textit{abort} \qquad \textit{and} \qquad \langle f \rangle; [P] \ = \ \textit{magic}.$$

Proof of Theorem 4:

$$\begin{aligned}
& \{\phi_1^{-1}\}; S_1 \ \vee \ \{\phi_2^{-1}\}; S_2 \\
= \quad & \qquad \{\text{Theorem 3 (separation)}\} \\
& \langle \phi_1 \rangle; (\{\phi_1^{-1}\}; S_1 \ \vee \ \{\phi_2^{-1}\}; S_2) \ \oplus \ \langle \phi_2 \rangle; (\{\phi_1^{-1}\}; S_1 \ \vee \ \{\phi_2^{-1}\}; S_2) \\
= \quad & \qquad \{\ \langle \phi_1 \rangle, \langle \phi_2 \rangle \text{ disjunctive}\} \\
& (\langle \phi_1 \rangle; \{\phi_1^{-1}\}; S_1 \ \vee \ \langle \phi_1 \rangle; \{\phi_2^{-1}\}; S_2) \ \oplus \ (\langle \phi_2 \rangle; \{\phi_1^{-1}\}; S_1 \ \vee \ \langle \phi_2 \rangle; \{\phi_2^{-1}\}; S_2) \\
= \quad & \qquad \{\text{partition property of } \phi_1, \phi_2 \text{ and Lemmas 1 and 2}\} \\
& (\textit{skip}; S_1 \ \vee \ \textit{abort}; S_2) \ \oplus \ (\textit{abort}; S_1 \ \vee \ \textit{skip}; S_2) \\
= \quad & \qquad \{\textit{abort}; S = \textit{abort}, \ \ S \vee \textit{abort} = S\} \\
& S_1 \oplus S_2.
\end{aligned}$$

$$
\begin{array}{ll}
 & [\phi_1^{-1}];S_1 \;\wedge\; [\phi_2^{-1}];S_2 \\
= & \langle\phi_1\rangle;([\phi_1^{-1}];S_1 \;\wedge\; [\phi_2^{-1}];S_2) \;\oplus\; \langle\phi_2\rangle;([\phi_1^{-1}];S_1 \;\wedge\; [\phi_2^{-1}];S_2) \\
= & \qquad \{\ \langle\phi_1\rangle, \langle\phi_2\rangle \text{ conjunctive}\} \\
 & (\langle\phi_1\rangle;[\phi_1^{-1}];S_1 \;\wedge\; \langle\phi_1\rangle;[\phi_2^{-1}];S_2) \;\oplus\; (\langle\phi_2\rangle;[\phi_1^{-1}];S_1 \;\wedge\; \langle\phi_2\rangle;[\phi_2^{-1}];S_2) \\
= & (skip;S_1 \;\wedge\; magic;S_2) \;\oplus\; (magic;S_1 \;\vee\; skip;S_2) \\
= & \qquad \{magic;S = magic,\ S \wedge magic = S\} \\
 & S_1 \oplus S_2.
\end{array}
$$

□

Theorem 4 allows us to derive properties of the summation operator from properties of angelic choice and demonic choice:

THEOREM **5** *The summation operator preserves monotonicity and bottom and top homomorphism, as well as conjunctivity, disjunctivity, and continuity.*

Proof: Preservation of monotonicity follows from Theorem 4 and the fact that the choice and sequencing constructs preserve monotonicity. Preservation of bottom homomorphism and disjunctivity follows from Theorem 4 and the fact that they are preserved by angelic choice and sequential composition. Preservation of top homomorphism and conjunctivity follows from Theorem 4 and the fact that they are preserved by demonic choice and sequential composition. □

Summation with *abort* and *magic* gives the following:

THEOREM **6** $S_1 \oplus abort = \{\phi_1^{-1}\};S_1$ *and* $S_1 \oplus magic = [\phi_1^{-1}];S_1$.

Proof:

$$
\begin{array}{rcl}
S_1 \oplus abort & = & \{\phi_1^{-1}\};S_1 \;\vee\; \{\phi_2^{-1}\};abort \\
 & = & \{\phi_1^{-1}\};S_1 \;\vee\; abort \\
 & = & \{\phi_1^{-1}\};S_1.
\end{array}
$$

$$
\begin{array}{rcl}
S_1 \oplus magic & = & [\phi_1^{-1}];S_1 \;\wedge\; [\phi_2^{-1}];magic \\
 & = & [\phi_1^{-1}];S_1 \;\wedge\; magic \\
 & = & [\phi_1^{-1}];S_1.
\end{array}
$$

□

Summation distributes through disjunction and conjunction as follows:

THEOREM **7**

(i) $(S_1 \vee S_1') \oplus S_2 = (S_1 \oplus S_2) \vee (S_1' \oplus S_2)$ *and* $S_1 \oplus (S_2 \vee S_2') = (S_1 \oplus S_2) \vee (S_1 \oplus S_2')$.

(ii) $(S_1 \wedge S_1') \oplus S_2 = (S_1 \oplus S_2) \wedge (S_1' \oplus S_2)$ *and* $S_1 \oplus (S_2 \wedge S_2') = (S_1 \oplus S_2) \wedge (S_1 \oplus S_2')$.

(iii) $(S_1 \vee S_1') \oplus (S_2 \vee S_2') = (S_1 \oplus S_2) \vee (S_1' \oplus S_2')$.

(iv) $(S_1 \wedge S_1') \oplus (S_2 \wedge S_2') = (S_1 \oplus S_2) \wedge (S_1' \oplus S_2')$.

Proof: We prove **(iii)** and **(iv)**; **(i)** and **(ii)** are similar:

$$
\begin{array}{rcl}
(S_1 \vee S_1') \oplus (S_2 \vee S_2') & = & \{\phi_1^{-1}\};(S_1 \vee S_1') \;\vee\; \{\phi_2^{-1}\};(S_2 \vee S_2') \\
 & = & \qquad \{\ \{\phi_1^{-1}\} \text{ disjunctive}\} \\
 & & \{\phi_1^{-1}\};S_1 \;\vee\; \{\phi_1^{-1}\};S_1' \;\vee\; \{\phi_2^{-1}\};S_2 \;\vee\; \{\phi_2^{-1}\};S_2' \\
 & = & \{\phi_1^{-1}\};S_1 \;\vee\; \{\phi_2^{-1}\};S_2 \;\vee\; \{\phi_1^{-1}\};S_1' \;\vee\; \{\phi_2^{-1}\};S_2' \\
 & = & (S_1 \oplus S_2) \vee (S_1' \oplus S_2).
\end{array}
$$

$$
\begin{aligned}
(S_1 \wedge S_1') \oplus (S_2 \wedge S_2') &= [\phi_1^{-1}]; (S_1 \wedge S_1') \;\wedge\; [\phi_2^{-1}]; (S_2 \wedge S_2') \\
&= \qquad \{\ [\phi_1^{-1}] \text{ conjunctive}\} \\
&\quad\; [\phi_1^{-1}]; S_1 \;\wedge\; [\phi_1^{-1}]; S_1' \;\wedge\; [\phi_2^{-1}]; S_2 \;\wedge\; [\phi_2^{-1}]; S_2' \\
&= [\phi_1^{-1}]; S_1 \;\wedge\; [\phi_2^{-1}]; S_2 \;\wedge\; [\phi_1^{-1}]; S_1' \;\wedge\; [\phi_2^{-1}]; S_2' \\
&= (S_1 \oplus S_2) \wedge (S_1' \oplus S_2).
\end{aligned}
$$

□

Part **(iii)** of Theorem 7 is interesting. By applying **(i)** twice, we get

$$(S_1 \vee S_1') \oplus (S_2 \vee S_2') \;=\; (S_1 \oplus S_2) \vee (S_1 \oplus S_2') \vee (S_1' \oplus S_2) \vee (S_1' \oplus S_2').$$

But, because of the type discrimination between Σ_1 and Σ_2, if the initial state is from Σ_1 then the effect of the first and second disjuncts of the right-hand side will be the same, and likewise if the initial state is in Σ_2. A similar observation may be made about part **(iv)**.

3.2 Derived Summation Operator

Next we define a summation operator on predicate transformers with distinct sources and distinct targets. Let ψ_1 and ψ_2 be the injections for the summation $\Gamma_1 + \Gamma_2$.

Definition **3** *For* $S_1 : \Sigma_1 \longmapsto \Gamma_1$, $\;S_2 : \Sigma_2 \longmapsto \Gamma_2$, $\;S_1 + S_2$ *is of type* $\Sigma_1 + \Sigma_2 \longmapsto \Gamma_1 + \Gamma_2$*, where*

$$S_1 + S_2 \;\;\widehat{=}\;\; S_1; \langle \psi_1 \rangle \;\oplus\; S_2; \langle \psi_2 \rangle.$$

Theorem **8** *The derived summation operator (Definition 3) preserves monotonicity and bottom and top homomorphism, as well as conjunctivity, disjunctivity, and continuity.*

Proof: Preservation of these properties follows from Theorem 5 and the fact that they are preserved by deterministic update and sequential composition. □

The components of $S_1 + S_2$ may be selected as follows:

Theorem **9** *For* $S_1 : \Sigma_1 \longmapsto \Gamma_1$, $S_2 : \Sigma_2 \longmapsto \Gamma_2$,

$$
\begin{aligned}
\langle \phi_1 \rangle; (S_1 + S_2); \{\psi_1^{-1}\} &= \langle \phi_1 \rangle; (S_1 + S_2); [\psi_1^{-1}] = S_1 \\
\langle \phi_2 \rangle; (S_1 + S_2); \{\psi_2^{-1}\} &= \langle \phi_2 \rangle; (S_1 + S_2); [\psi_2^{-1}] = S_2.
\end{aligned}
$$

Proof:

$$
\begin{aligned}
\langle \phi_1 \rangle; (S_1 + S_2); \{\psi_1^{-1}\} &= \qquad \{\text{Definition 3}\} \\
&\quad\; \langle \phi_1 \rangle; (S_1; \langle \psi_1 \rangle \;\oplus\; S_2; \langle \psi_1 \rangle); \{\psi_1^{-1}\} \\
&= \qquad \{\text{Theorem 3}\} \\
&\quad\; S_1; \langle \psi_1 \rangle; \{\psi_1^{-1}\} \\
&= \qquad \{\text{Lemma 1}\} \\
&\quad\; S_1.
\end{aligned}
$$

Similarly for the other equalities. □

The derived summation operator does not, in general, satisfy a separation property, i.e., for $S : \Sigma_1 + \Sigma_2 \longmapsto \Gamma_1 + \Gamma_2$,

$$S \;\neq\; \langle \phi_1 \rangle; S; \{\psi_1^{-1}\} \;+\; \langle \phi_2 \rangle; S; \{\psi_2^{-1}\}.$$

For example, consider the following constant update statement (let $\gamma_1 : \Gamma_1$ be some fixed value):

$$\langle \lambda \sigma : \Sigma_1 + \Sigma_2 \cdot \psi_1\ \gamma_1 \rangle : \Sigma_1 + \Sigma_2 \;\longmapsto\; \Gamma_1 + \Gamma_2.$$

We have

$$
\begin{array}{ll}
 & \langle \phi_1 \rangle ; \langle \lambda\sigma : \Sigma_1 + \Sigma_2 \cdot \psi_1\ \gamma_1 \rangle ; \{\psi_1^{-1}\} \quad + \quad \langle \phi_2 \rangle ; \langle \lambda\sigma : \Sigma_1 + \Sigma_2 \cdot \psi_1\ \gamma_1 \rangle ; \{\psi_2^{-1}\} \\
= & \qquad \{\text{calculation}\} \\
 & \langle \lambda\sigma_1 : \Sigma_1 \cdot \gamma_1 \rangle ; \langle \psi_1 \rangle ; \{\psi_1^{-1}\} \quad + \quad \langle \lambda\sigma_2 : \Sigma_2 \cdot \gamma_1 \rangle ; \langle \psi_1 \rangle ; \{\psi_2^{-1}\} \\
= & \qquad \{\text{Lemmas 1 and 2}\} \\
 & \langle \lambda\sigma_1 : \Sigma_1 \cdot \gamma_1 \rangle \ + \ abort \\
\neq & \langle \lambda\sigma : \Sigma_1 + \Sigma_2 \cdot \psi_1\ \gamma_1 \rangle .
\end{array}
$$

The reason that $\langle \lambda x : \Sigma_1 + \Sigma_2 \cdot \psi_1\ \gamma_1 \rangle$ cannot be separated is that it always ends in a state in Γ_1 even if the initial state is in Σ_2.

The following properties follow from the corresponding properties of the $\oplus$ operator:

Theorem 10

(i) $S_1 + abort = \{\phi_1^{-1}\}; S_1; \langle \psi_1 \rangle$.

(ii) $S_1 + magic = [\phi_1^{-1}]; S_1; \langle \psi_1 \rangle$.

(iii) $(S_1 \vee S_1') + S_2 = (S_1 + S_2) \vee (S_1' + S_2)$ *and* $S_1 + (S_2 \vee S_2') = (S_1 + S_2) \vee (S_1 + S_2')$.

(iv) $(S_1 \wedge S_1') + S_2 = (S_1 + S_2) \wedge (S_1' + S_2)$ *and* $S_1 + (S_2 \wedge S_2') = (S_1 + S_2) \wedge (S_1 + S_2')$.

(v) $(S_1 \vee S_1') + (S_2 \vee S_2') = (S_1 + S_2) \vee (S_1' + S_2')$.

(vi) $(S_1 \wedge S_1') + (S_2 \wedge S_2') = (S_1 + S_2) \wedge (S_1' + S_2')$.

Summation distributes through sequential composition as follows:

Theorem 11

(i) $(S_1 + S_2)\ ;\ (T_1 \oplus T_2) \quad = \quad (S_1; T_1) \oplus (S_2; T_2)$.

(ii) $(S_1 + S_2)\ ;\ (T_1 + T_2) \quad = \quad (S_1; T_1) + (S_2; T_2)$.

Proof Firstly (i):

$$
\begin{array}{lll}
(S_1 + S_2)\ ;\ (T_1 \oplus T_2) & = & \qquad \{\text{Theorem 4 and Definition 3}\} \\
 & & ([\phi_1^{-1}]; S_1; \langle \psi_1 \rangle \ \wedge \ [\phi_2^{-1}]; S_2; \langle \psi_2 \rangle) \ \ ; \ \ (T_1 \oplus T_2) \\
 & = & \qquad \{\text{Distribution of } \wedge \text{ and } ;\} \\
 & & [\phi_1^{-1}]; S_1; \langle \psi_1 \rangle; (T_1 \oplus T_2) \ \ \wedge \ \ [\phi_2^{-1}]; S_2; \langle \psi_2 \rangle; (T_1 \oplus T_2) \\
 & = & \qquad \{\text{Theorem 3}\} \\
 & & [\phi_1^{-1}]; S_1; T_1 \ \ \wedge \ \ [\phi_2^{-1}]; S_2; T_2 \\
 & = & \qquad \{\text{Theorem 4}\} \\
 & & S_1; T_1 \ \oplus \ S_2; T_2 .
\end{array}
$$

(ii) then follows from (i). □

In order to consider the summation of angelic and demonic updates, we require the following definition of summation of relations:

Definition 4 *For* $P_1 : \Sigma_1 \leftrightarrow \Gamma_1,\ P_2 : \Sigma_2 \leftrightarrow \Gamma_2,\ \sigma : \Sigma_1 + \Sigma_2,\ \gamma : \Gamma_1 + \Gamma_2$,

$$
\begin{array}{lll}
(P_1 + P_2)\ \sigma\ \gamma & \hat{=} & \sigma \in (dom\, \phi_1^{-1}) \ \wedge \ \gamma \in (dom\, \psi_1^{-1}) \ \wedge \ P_1\ (\phi_1^{-1}\sigma)\ (\psi_1^{-1}\gamma) \quad \vee \\
 & & \sigma \in (dom\, \phi_2^{-1}) \ \wedge \ \gamma \in (dom\, \psi_2^{-1}) \ \wedge \ P_2\ (\phi_2^{-1}\sigma)\ (\psi_2^{-1}\gamma).
\end{array}
$$

We have the following lemma:

Lemma 3 *For relations* $P_1 : \Sigma_1 \leftrightarrow \Gamma_1,\ P_2 : \Sigma_2 \leftrightarrow \Gamma_2$,

$$
P_1 + P_2 \quad = \quad \phi_1^{-1}; P_1; \psi_1 \ \vee \ \phi_2^{-1}; P_1; \psi_2 .
$$

Proof: see appendix.

Note that summation of relations may be characterised only as a disjunction of relations, whereas summation of predicate transformers may be characterised in both conjunctive and disjunctive form.

Now, summation distributes through angelic and demonic update as follows:

THEOREM **12** *For relations* $P_1 : \Sigma_1 \leftrightarrow \Gamma_1$, $P_2 : \Sigma_2 \leftrightarrow \Gamma_2$,

$$\{P_1\} + \{P_2\} = \{P_1 + P_2\} \quad \textit{and} \quad [P_1] + [P_2] = [P_1 + P_2]. \tag{4}$$

Proof:

$$\begin{array}{cl} \{P_1\} + \{P_2\} = & \quad \{\text{Theorem 4}\} \\ & \{\phi_1^{-1}\}; \{P_1\}; \langle\psi_1\rangle \ \vee \ \{\phi_2^{-1}\}; \{P_2\}; \langle\psi_2\rangle \\ = & \quad \{\text{Properties of angelic choice and update}\} \\ & \{\phi_1^{-1}; P_1; \psi_1 \ \vee \ \phi_2^{-1}; P_2; \psi_2\} \\ = & \quad \{\text{Lemma 3}\} \\ & \{P_1 + P_2\}. \end{array}$$

$$\begin{array}{cl} [P_1] + [P_2] = & \quad \{\text{Theorem 4}\} \\ & [\phi_1^{-1}]; [P_1]; \langle\psi_1\rangle \ \wedge \ [\phi_2^{-1}]; [P_2]; \langle\psi_2\rangle \\ = & \quad \{\text{Properties of demonic choice and update}\} \\ & [\phi_1^{-1}; P_1; \psi_1 \ \vee \ \phi_2^{-1}; P_2; \psi_2] \\ = & \quad \{\text{Lemma 3}\} \\ & [P_1 + P_2]. \end{array}$$

□

We have the following lemma about the summation of test relations:

LEMMA **4** *For predicates* $p_1 : \mathcal{P}\,\Sigma_1$, $p_2 : \mathcal{P}\,\Sigma_2$,

$$\overline{p_1} + \overline{p_2} \quad = \quad \overline{p_1 + p_2}.$$

Proof: see appendix.

From this lemma, and the distribution of summation through angelic and demonic choice, we have that summation distributes through assert and guard statements:

THEOREM **13** *For predicates* $p_1 : \mathcal{P}\,\Sigma_1$, $p_2 : \mathcal{P}\,\Sigma_2$,

$$\{p_1\} + \{p_2\} = \{p_1 + p_2\} \quad \textit{and} \quad [p_1] + [p_2] = [p_1 + p_2].$$

3.3 Refinement

Both the summation operators preserve refinement allowing us to refine summands separately:

THEOREM **14**

$$\begin{array}{lcl} S_1 \leq S_1' \wedge S_2 \leq S_2' & \Rightarrow & (S_1 \oplus S_2) \leq (S_1' \oplus S_2') \\ T_1 \leq T_1' \wedge T_2 \leq T_2' & \Rightarrow & (T_1 + T_2) \leq (T_1' + T_2'). \end{array}$$

Proof of this follows directly from Theorem 4 and the fact that sequential composition and the choice operators preserve refinement.

The well-known technique of data refinement involves replacing abstract program variables with concrete program variables using an abstraction relation [11]. In the refinement calculus, the abstraction relation is modelled by an abstraction command, and we say that $S : \Delta(\Sigma)$ is data refined by $S' : \Delta(\Sigma')$ under abstraction command $\alpha : \Sigma' \longmapsto \Sigma$ if $\alpha; S \ \leq \ S'; \alpha$ [9, 26]. We can easily show that summation preserves data refinement:

THEOREM **15** *Let* $S_1 : \Delta(\Sigma_1)$, $S'_1 : \Delta(\Sigma'_1)$, $S_2 : \Delta(\Sigma_2)$, $S'_2 : \Delta(\Sigma'_2)$, *be predicate transformers. Let* $\alpha_1 : \Sigma'_1 \longmapsto \Sigma_1$, $\alpha_2 : \Sigma'_2 \longmapsto \Sigma_2$ *be abstraction commands. Then*

$$\begin{array}{l} \alpha_1; S_1 \leq S'_1; \alpha_1 \quad \wedge \quad \alpha_2; S_2 \leq S'_2; \alpha_2 \ \Rightarrow \\ \qquad (\alpha_1 + \alpha_2); (S_1 + S_2) \ \leq \ (S'_1 + S'_2); (\alpha_1 + \alpha_2). \end{array}$$

Proof:

$$\begin{array}{rcl} (\alpha_1 + \alpha_2); (S_1 + S_2) & = & \qquad \{\text{Distribution of } + \text{ and } ;\} \\ & & (\alpha_1; S_1) + (\alpha_2; S_2) \\ & \leq & \qquad \{\text{Theorem 14 and assumptions}\} \\ & & (S'_1; \alpha_1) + (S'_2; \alpha_2) \\ & = & (S'_1 + S'_2); (\alpha_1 + \alpha_2). \end{array}$$

□

3.4 Recursion

A recursive command has the form $(\mu X \cdot F\ X)$ (least fixed-point of F) where X ranges over $\Delta(\Sigma)$ and F is a function from $\Delta(\Sigma)$ to $\Delta(\Sigma)$. F should be monotonic (w.r.t. refinement) for the least fixed-point to exist. For example, the DO-loop is defined recursively as follows:

$$\mathbf{do}\ S\ \mathbf{od} \quad = \quad (\mu X \cdot S; X \quad \wedge \quad [\neg\, gd\ S]\ ).$$

Summation distributes through recursion as follows:

THEOREM **16** *Given* $F_1 : \Delta(\Sigma_1) \rightarrow \Delta(\Sigma_1)$, $F_2 : \Delta(\Sigma_2) \rightarrow \Delta(\Sigma_2)$, *let*

$$F\ Y \quad \widehat{=} \quad F_1(\langle\phi_1\rangle; Y; \{\phi_1^{-1}\}) \ + \ F_2(\langle\phi_2\rangle; Y; \{\phi_2^{-1}\}).$$

Then

$$(\mu X \cdot F_1 X) \ + (\mu X \cdot F_2 X) \quad = \quad (\mu Y \cdot F\ Y).$$

In order to prove Theorem 16 on recursion, we require the following *Generalised Limit Theorem* which does not depend on f being continuous:

THEOREM **17** *Let* f *be a monotonic function and for ordinal* α *let* f^α *be defined inductively by*

$$f^\alpha \quad = \quad (\vee \beta < \alpha \cdot f\ f^\beta).$$

Then f *has a least fixed-point given by* f^α *for some ordinal* α.

This version of the theorem is from [23] and is based on [12].

Proof of Theorem 16: Firstly, we have

$$\begin{array}{rcl} F\ (F_1^\beta + F_2^\beta) & = & F_1(\langle\phi_1\rangle; (F_1^\beta + F_2^\beta); \{\phi_1^{-1}\}) \ + \ F_2(\langle\phi_2\rangle; (F_1^\beta + F_2^\beta); \{\phi_2^{-1}\}) \\ & = & \qquad \{\text{Theorem 9 (Selection)}\} \\ & & F_1\ F_1^\beta \ + \ F_2\ F_2^\beta. \end{array}$$

Next, assume $F^\beta = F_1^\beta + F_2^\beta$, for $\beta < \alpha$, then

$$\begin{array}{rcl} F^\alpha & = & (\vee \beta < \alpha \cdot F\ F^\beta) \\ & = & \qquad \{\text{Assumption}\} \\ & & (\vee \beta < \alpha \cdot F\ (F_1^\beta + F_2^\beta)) \\ & = & (\vee \beta < \alpha \cdot F_1\ F_1^\beta \ + \ F_2\ F_2^\beta) \\ & = & \qquad \{\text{Theorem 4}\} \\ & & (\vee \beta < \alpha \cdot \{\phi_1^{-1}\}; (F_1\ F_1^\beta); \langle\phi_1\rangle \quad \vee \quad \{\phi_2^{-1}\}; (F_2\ F_2^\beta); \langle\phi_2\rangle) \\ & = & (\vee \beta < \alpha \cdot \{\phi_1^{-1}\}; (F_1\ F_1^\beta); \langle\phi_1\rangle) \ \vee \ (\vee \beta < \alpha \cdot \{\phi_2^{-1}\}; (F_2\ F_2^\beta); \langle\phi_2\rangle) \\ & = & \qquad \{\ \{\phi_1^{-1}\} \text{ and } \{\phi_2^{-1}\} \text{ disjunctive}\} \\ & & \{\phi_1^{-1}\}; (\vee \beta < \alpha \cdot F_1\ F_1^\beta); \langle\phi_1\rangle \ \vee \ \{\phi_2^{-1}\}; (\vee \beta < \alpha \cdot F_2\ F_2^\beta); \langle\phi_2\rangle \\ & = & \{\phi_1^{-1}\}; F_1^\alpha; \langle\phi_1\rangle \ \vee \ \{\phi_2^{-1}\}; F_2^\alpha; \langle\phi_2\rangle \\ & = & F_1^\alpha \ + \ F_2^\alpha. \end{array}$$

Hence, $F^\alpha = F_1^\alpha \ + \ F_2^\alpha$, for all α. Finally, It is easy to show that there is an α such that $F_1(F_1^\alpha) = F_1^\alpha$, $F_2(F_2^\alpha) = F_2^\alpha$, and $F(F^\alpha) = F^\alpha$ (simply choose an α greater than the cardinality of $\Sigma_1 + \Sigma_2$), thus,

$$(\mu X \cdot F\ X) \quad = \quad (\mu X \cdot F_1 X) \ + (\mu X \cdot F_2 X).$$

□

We shall attempt to show that summation distributes through DO-loops:

$$\begin{array}{ll}
& \textbf{do } S_1 \textbf{ od} \ \ + \ \ \textbf{do } S_2 \textbf{ od} \\
= & (\mu X \cdot S_1; X \ \ \wedge \ \ [\neg\, gd\ S_1]\ ) \ \ + \ \ (\mu X \cdot S_2; X \ \ \wedge \ \ [\neg\, gd\ S_2]\ ) \\
= & \qquad \{\text{distribution of + through recursion}\} \\
& (\mu Y \cdot (S_1; \langle \phi_1 \rangle; Y; \{\phi_1^{-1}\} \ \ \wedge \ \ [\neg\, gd\ S_1]\ ) \ \ + \ \ (S_2; \langle \phi_2 \rangle; Y; \{\phi_2^{-1}\} \ \ \wedge \ \ [\neg\, gd\ S_2]\ )\ ) \\
= & \qquad \{\text{distribution of + and } \wedge\} \\
& (\mu Y \cdot (S_1; \langle \phi_1 \rangle; Y; \{\phi_1^{-1}\} \ + \ S_2; \langle \phi_2 \rangle; Y; \{\phi_2^{-1}\}) \ \wedge \ (\ [\neg\, gd\ S_1] \ + [\neg\, gd\ S_2]\ )\ ) \\
= & \qquad \{\text{distribution of + and ;}\} \\
& (\mu Y \cdot ((S_1 + S_2); (\langle \phi_1 \rangle; Y; \{\phi_1^{-1}\} \ + \ \langle \phi_2 \rangle; Y; \{\phi_2^{-1}\})) \ \wedge \ (\ [\neg\, gd\ S_1] \ + [\neg\, gd\ S_2]\ )\ ) \\
= & \qquad \{ \ \ \begin{array}[t]{lll} [\neg\, gd\ S_1] \ + [\neg\, gd\ S_2] & = & [S_1 \perp] \ + [S_2 \perp] \\ & = & [S_1 \perp \ + \ S_2 \perp] \\ & = & [(S_1 + S_2) \perp] \\ & = & [\neg\, gd\ (S_1 + S_2)] \quad \} \end{array} \\
& (\mu Y \cdot (S_1 + S_2); (\langle \phi_1 \rangle; Y; \{\phi_1^{-1}\} \ + \ \langle \phi_2 \rangle; Y; \{\phi_2^{-1}\})) \ \wedge \ [\neg\, gd\ (S_1 + S_2)]\ )
\end{array}$$

We would like to be able to continue as follows:

$$\begin{array}{ll}
= & (\mu Y \cdot (S_1 + S_2); Y \ \wedge \ [\neg\, gd\ (S_1 + S_2)]\ ) \\
= & \textbf{do } S_1 + S_2 \textbf{ od}.
\end{array}$$

But it is not always the case that

$$Y \quad = \quad \langle \phi_1 \rangle; Y; \{\phi_1^{-1}\} \ + \ \langle \phi_2 \rangle; Y; \{\phi_2^{-1}\},$$

as the counterexample after Theorem 9 shows. However, we do have the following theorem:

THEOREM **18** *Let F be as defined in Theorem 16. If $F\ Y \ = \ G(\langle \phi_1 \rangle; Y; \{\phi_1^{-1}\} \ + \ \langle \phi_2 \rangle; Y; \{\phi_2^{-1}\})$, then*

$$(\mu Y \cdot F\ Y) \quad = \quad (\mu Y \cdot G\ Y).$$

Proof: We prove inductively that $F^\alpha = G^\alpha$:

$$\begin{array}{lll}
F^\alpha & = & (\vee \beta < \alpha \cdot F\ F^\beta) \\
& = & \qquad \{\text{antecedent}\} \\
& & (\vee \beta < \alpha \cdot G(\langle \phi_1 \rangle; F^\beta; \{\phi_1^{-1}\} \ + \ \langle \phi_2 \rangle; F^\beta; \{\phi_2^{-1}\})) \\
& = & \qquad \{ \ \ \begin{array}[t]{ll} & \langle \phi_1 \rangle; F^\beta; \{\phi_1^{-1}\} \ + \ \langle \phi_2 \rangle; F^\beta; \{\phi_2^{-1}\} \\ = & \langle \phi_1 \rangle; (F_1^\beta + F_2^\beta); \{\phi_1^{-1}\} \ + \ \langle \phi_2 \rangle; (F_1^\beta + F_2^\beta); \{\phi_2^{-1}\} \\ = & F_1^\beta \ + \ F_2^\beta \\ = & F^\beta \qquad \} \end{array} \\
& & (\vee \beta < \alpha \cdot G\ F^\beta) \\
& = & \qquad \{\text{induction hypothesis}\} \\
& & (\vee \beta < \alpha \cdot G\ G^\beta) \\
& = & G^\alpha.
\end{array}$$

$(\mu Y \cdot F\ Y) = (\mu Y \cdot G\ Y)$ then follows by the Generalised Limit Theorem. □

Now, by taking

$$\begin{array}{lll}
F\ Y & = & (S_1; \langle \phi_1 \rangle; Y; \{\phi_1^{-1}\} \ \ \wedge \ \ [\neg\, gd\ S_1]\ ) \ \ + \ \ (S_2; \langle \phi_2 \rangle; Y; \{\phi_2^{-1}\} \ \ \wedge \ \ [\neg\, gd\ S_2]\ ), \\
G\ Y & = & (S_1 + S_2); Y \ \wedge \ [\neg\, gd\ (S_1 + S_2)],
\end{array}$$

the step in the above derivation of **do** $S_1 + S_2$ **od** where $\langle\phi_1\rangle; Y; \{\phi_1^{-1}\} \ + \ \langle\phi_2\rangle; Y; \{\phi_2^{-1}\}$ is replaced by Y is justified, giving:

THEOREM **19** $\quad$ **do** S_1 **od** $\ + \ $ **do** S_2 **od** $\ = \ $ **do** $S_1 + S_2$ **od**.

4 Product Operators

The cartesian product of state spaces Σ_1 and Σ_2 is denoted $\Sigma_1 \times \Sigma_2$. Given two predicates q_1 and q_2 their product is denoted $q_1 \times q_2$ and is defined as follows:

DEFINITION **5** *For predicates* $q_1 : \mathcal{P}\,\Sigma_1$, $q_2 : \mathcal{P}\,\Sigma_2$, *predicate* $q_1 \times q_2$ *is of type* $\mathcal{P}\,(\Sigma_1 \times \Sigma_2)$ *where for* $\sigma_1 : \Sigma_1, \quad \sigma_2 : \Sigma_2$,

$$(q_1 \times q_2)\ (\sigma_1, \sigma_2) \quad = \quad (q_1\ \sigma_1) \wedge (q_2\ \sigma_2).$$

Note that, in contrast to summation of predicates where any predicate of type $\mathcal{P}\,(\Sigma_1 + \Sigma_2)$ may be represented by the summation of two predicates of types $\mathcal{P}\,\Sigma_1$ and $\mathcal{P}\,\Sigma_2$ respectively, predicates of the form $q_1 \times q_2$ only form a rectangular subset of $\mathcal{P}\,(\Sigma_1 \times \Sigma_2)$. Also, the components of a product of predicates cannot always be selected, e.g., p_1 cannot be retrieved from $p_1 \times \perp$. As we shall see, these facts mean that the product operators for predicate transformers do not satisfy as many properties as the summation operator.

Consider the following two predicate transformers which have a common source but distinct targets:

$$S_1 : \Sigma \longmapsto \Gamma_1, \qquad S_2 : \Sigma \longmapsto \Gamma_2.$$

The product operator combines these two commands to form a command with target $\Gamma_1 \times \Gamma_2$ as follows:

DEFINITION **6** *For* $S_1 : \Sigma \longmapsto \Gamma_1$, $\ S_2 : \Sigma \longmapsto \Gamma_2$, $\ S_1 \otimes S_2$ *is of type* $\Sigma \longmapsto \Gamma_1 \times \Gamma_2$, *where for* $q : \mathcal{P}\,(\Gamma_1 \times \Gamma_2)$:

$$(S_1 \otimes S_2)\ q \quad \widehat{=} \quad (\exists q_1 : \mathcal{P}\,\Gamma_1;\ q_2 : \mathcal{P}\,\Gamma_2 \,|\, q_1 \times q_2 \le q \cdot S_1 q_1 \ \wedge \ S_2 q_2).$$

S_1 and S_2 are commands that read from the same initial state space, but write to separate final state spaces. The product of S_1 and S_2 reads from Σ and writes to $\Gamma_1 \times \Gamma_2$, and we shall see later that execution of $S_1 \otimes S_2$ has the same effect as simultaneous execution of S_1 and S_2.

Before investigating properties of the product operator further, we first look at two related operators. The first is derived from the product operator. Let π_1 and π_2 be the projections from $\Sigma_1 \times \Sigma_2$ to Σ_1 and Σ_2 respectively:

$$\begin{array}{ll} \pi_1 : \Sigma_1 \times \Sigma_2 \to \Sigma_1, & \pi_2 : \Sigma_1 \times \Sigma_2 \to \Sigma_2, \\ \pi_1(\sigma_1, \sigma_2) = \sigma_1, & \pi_2(\sigma_1, \sigma_2) = \sigma_2. \end{array}$$

The projections give the following deterministic updates:

$$\langle\pi_1\rangle : \Sigma_1 \times \Sigma_2 \longmapsto \Sigma_1 \quad \text{and} \quad \langle\pi_2\rangle : \Sigma_1 \times \Sigma_2 \longmapsto \Sigma_2.$$

The statement $\langle\pi_1\rangle$ starts in a state (σ_1, σ_2) and ends in the state σ_1 simply discarding the second component. The derived product operator combines commands with distinct sources and distinct targets, and is defined as follows:

DEFINITION **7** *For* $S_1 : \Sigma_1 \longmapsto \Gamma_1$, $\ S_2 : \Sigma_2 \longmapsto \Gamma_2$, $\ S_1 \times S_2$ *is of type* $\Sigma_1 \times \Sigma_2 \longmapsto \Gamma_1 \times \Gamma_2$, *where*

$$S_1 \times S_2 \quad \widehat{=} \quad \langle\pi_1\rangle; S_1 \ \otimes \ \langle\pi_2\rangle; S_2.$$

We shall see later that the derived product operator models simultaneous execution of two commands with distinct initial states and distinct final states.

Definition 6 may be generalised to define what we term a *fusion* operator that combines commands with common sources and common targets as follows:

DEFINITION **8** *For* $S_1, S_2 : \Sigma \longmapsto \Gamma$, $\ S_1 \odot S_2$ *is of type* $\Sigma \longmapsto \Gamma$, *where for* $q : \mathcal{P}\,\Gamma$:

$$(S_1 \odot S_2)\ q \quad \widehat{=} \quad (\exists q_1, q_2 : \mathcal{P}\,\Gamma \,|\, q_1 \wedge q_2 \le q \cdot S_1 q_1 \ \wedge \ S_2 q_2).$$

We proceed by investigating properties of the fusion operator and showing that the product operators are special cases of the fusion operator.

4.1 Properties of Fusion Operator

THEOREM **20** *The fusion operator (Definition 8) preserves monotonicity, top homomorphism, and conjunctivity.*

Proof: see appendix.

THEOREM **21** *If either S_1 or S_2 is disjunctive then $S_1 \odot S_2$ is disjunctive. Similarly for continuity.*

Proof see appendix.

This theorem tells us that if either S_1 or S_2 is deterministic, then the fusion of S_1 and S_2 is deterministic, and if either S_1 or S_2 is boundedly nondeterministic, then the fusion of S_1 and S_2 is boundedly nondeterministic. Note that the fusion operator does not preserve bottom homomorphism. For example, although both the deterministic updates $\langle \lambda x \cdot 0\rangle$ and $\langle \lambda x \cdot 1\rangle$ are bottom homomorphic, we can easily show that

$$\langle \lambda x \cdot 0\rangle \odot \langle \lambda x \cdot 1\rangle \quad = \quad \mathit{magic},$$

which is certainly not bottom homomorphic.

The fusion operator combines with *abort* and *magic* as follows:

THEOREM **22** *For predicate transformer S, and monotonic predicate transformer T,*

(i) $S \odot \mathit{abort} = \mathit{abort}$

(ii) $T \odot \mathit{magic} = \{T\ \top\}; \mathit{magic}.$

Proof: (i) is trivial. For (ii), we have:

$$\begin{array}{rcl}
(T \odot \mathit{magic})\ q & = & (\exists q_1, q_2 : \mathcal{P}\,\Gamma \mid q_1 \wedge q_2 \leq q \cdot T\ q_1\ \wedge\ \mathit{magic}\ q_2) \\
& = & (\exists q_1, q_2 : \mathcal{P}\,\Gamma \mid q_1 \wedge q_2 \leq q \cdot T\ q_1) \\
& = & (\exists q_1 : \mathcal{P}\,\Gamma \cdot T\ q_1) \\
& = & \qquad \{T \text{ monotonic}\} \\
& & T\ \top \\
& = & \{T\ \top\}; \mathit{magic}\ q.
\end{array}$$

□

The *havoc* statement, where $\mathit{havoc} = [\lambda \sigma, \gamma \cdot \mathit{true}]$, is the most (demonically) nondeterministic statement that is guaranteed to terminate. We have that *havoc* acts as a unit for the fusion operator:

THEOREM **23** *For monotonic predicate transformer S,*

$$S \odot \mathit{havoc} = S.$$

Proof: See appendix.

The fusion operator distributes through disjunction and conjunction:

THEOREM **24**

(i) $(S_1 \vee S_1') \odot S_2 = (S_1 \odot S_2) \vee (S_1' \odot S_2)$ *and* $S_1 \odot (S_2 \vee S_2') = (S_1 \odot S_2) \vee (S_1 \odot S_2')$.

(ii) *for monotonic S_1, S_1', conjunctive S_2 (and vice versa):*
$(S_1 \wedge S_1') \odot S_2 = (S_1 \odot S_2) \wedge (S_1' \odot S_2)$ *and* $S_1 \odot (S_2 \wedge S_2') = (S_1 \odot S_2) \wedge (S_1 \odot S_2')$.

Proof: Distribution through disjunction is trivial. See appendix for distribution through conjunction.

The following theorem illustrates that the effect of the fusion operator is to reduce the (demonic) nondeterminism of the terminating behaviour of both commands:

THEOREM **25** *For relations $P, Q : \Sigma \leftrightarrow \Gamma$, and predicates $p, q : \mathcal{P}\,\Sigma$,*

$$\{p\}; [P]\ \odot\ \{q\}; [Q] \quad = \quad \{p \wedge q\}; [P \wedge Q].$$

Proof: See appendix.

The fusion operator preserves refinement:

THEOREM **26** $\quad S_1 \leq S_1' \ \wedge\ S_2 \leq S_2' \quad \Rightarrow \quad (S_1 \odot S_2) \leq (S_1' \odot S_2')$.

Proof:

$$\begin{array}{rcl}
(S_1 \odot S_2)\ q & = & (\exists q_1, q_2 : \mathcal{P}\,\Gamma \mid q_1 \wedge q_2 \leq q \cdot S_1 q_1 \ \wedge\ S_2 q_2) \\
& \leq & \qquad \{\text{assumption}\} \\
& & (\exists q_1, q_2 : \mathcal{P}\,\Gamma \mid q_1 \wedge q_2 \leq q \cdot S_1' q_1 \ \wedge\ S_2' q_2) \\
& = & (S_1' \odot S_2')\ q.
\end{array}$$

□

The fusion operator acts as a form of 'co-refinement' operator:

THEOREM **27** *For predicate transformers S_1 and S_2, $S_1 \odot S_2$ solves:*

$$\{S_2 \top\}; S_1 \quad \leq \quad X \tag{5}$$
$$\{S_1 \top\}; S_2 \quad \leq \quad X, \tag{6}$$

and furthermore, for any conjunctive Y that solves (5) and (6),

$$S_1 \odot S_2 \quad \leq \quad Y. \tag{7}$$

Proof: Firstly, $S_1 \odot S_2$ solves (5):

$$\begin{array}{rcl}
(S_1 \odot S_2)\ q & = & (\exists q_1, q_2 : \mathcal{P}\,\Gamma \mid q_1 \wedge q_2 \leq q \cdot S_1 q_1 \ \wedge\ S_2 q_2) \\
& \geq & S_1 q \ \wedge\ S_2 \top \\
& = & \{S_2 \top\}; S_1\ q.
\end{array}$$

Similarly, $S_1 \odot S_2$ solves (6). Finally, $S_1 \odot S_2$ satisfies (7):

$$\begin{array}{rcl}
(S_1 \odot S_2)\ q & = & (\exists q_1, q_2 : \mathcal{P}\,\Gamma \mid q_1 \wedge q_2 \leq q \cdot S_1 q_1 \ \wedge\ S_2 q_2) \\
& \leq & \qquad \{Y \text{ satisfies (5) and (6)}\} \\
& & (\exists q_1, q_2 : \mathcal{P}\,\Gamma \mid q_1 \wedge q_2 \leq q \cdot Y q_1 \ \wedge\ Y q_2) \\
& = & \qquad \{Y \text{ conjunctive}\} \\
& & (\exists q_1, q_2 : \mathcal{P}\,\Gamma \mid q_1 \wedge q_2 \leq q \cdot Y(q_1 \wedge q_2)) \\
& \leq & \qquad \{Y \text{ monotonic}\} \\
& & Y\ q.
\end{array}$$

□

Thus for conjunctive S_1 and S_2, $S_1 \odot S_2$ is the least conjunctive predicate transformer satisfying (5) and (6). This theorem allows us to prove the following absorption rule:

THEOREM **28** *For conjunctive S_1, S_2,*

$$S_1 \leq S_2 \quad \Rightarrow \quad S_1 \odot S_2 = S_2.$$

Proof: S_2 is the least conjunctive predicate transformer satisfying (5) and (6). □

Morgan [17] has developed an operator $\Box$ on predicate transformers such that $\Box S$ is the least-refined predicate transformer that is both universally conjunctive and refines S. By these properties, it is easy to show for universally conjunctive S_1 and S_2, that $\Box(S_1 \vee S_2)$ is the least conjunctive predicate transformer satisfying (5) and (6). Thus by Theorem 27 we have for universally conjunctive S_1 and S_2,

$$\Box(S_1 \vee S_2) \quad = \quad S_1 \odot S_2.$$

4.2 Properties of Product Operator

Now we show that the product operator is a special case of the fusion operator. Let ϖ_1 and ϖ_2 be the projections from $\Gamma_1 \times \Gamma_2$ to Γ_1 and Γ_2 respectively:

$$\varpi_1 : \Gamma_1 \times \Gamma_2 \rightarrow \Gamma_1, \qquad \varpi_2 : \Gamma_1 \times \Gamma_2 \rightarrow \Gamma_2.$$

We have the following lemma:

Lemma 5 *For* $p_1 : \mathcal{P}\,\Gamma_1,\ p_2 : \mathcal{P}\,\Gamma_2$,

$$\langle\varpi_1\rangle p_1 \ \wedge\ \langle\varpi_2\rangle p_2 \quad = \quad p_1 \times p_2.$$

Proof: For $\gamma_1 : \Gamma_1, \gamma_2 : \Gamma_2$,

$$\begin{aligned}
(\langle\varpi_1\rangle p_1 \wedge \langle\varpi_2\rangle p_2)\ (\gamma_1,\gamma_2) &= (\langle\varpi_1\rangle\ p_1\ (\gamma_1,\gamma_2)) \wedge (\langle\varpi_2\rangle\ p_2\ (\gamma_1,\gamma_2)) \\
&= (p_1\ (\varpi_1(\gamma_1,\gamma_2))) \wedge (p_2\ (\varpi_2(\gamma_1,\gamma_2))) \\
&= (p_1\ \gamma_1) \wedge (p_2\ \gamma_2) \\
&= (p_1 \times p_2)\ (\gamma_1,\gamma_2).
\end{aligned}$$

$\square$

The inverses of ϖ_1 and ϖ_2 are relations:

$$\varpi_1^{-1} : \Gamma_1 \leftrightarrow \Gamma_1 \times \Gamma_2, \qquad \varpi_2^{-1} : \Gamma_2 \leftrightarrow \Gamma_1 \times \Gamma_2,$$

so that the demonic update $[\varpi_1^{-1}]$ transforms predicates in $\Gamma_1 \times \Gamma_2$ to predicates in Γ_1:

$$[\varpi_1^{-1}] : \Gamma_1 \longmapsto \Gamma_1 \times \Gamma_2, \qquad (=\ \mathcal{P}\,(\Gamma_1 \times \Gamma_2) \rightarrow \mathcal{P}\,\Gamma_1),$$

and similarly for $[\varpi_2^{-1}]$. The command $[\varpi_1^{-1}]$ starts in a state $\gamma_1 : \Gamma_1$ and ends in a state $(\gamma_1', \gamma_2') : \Gamma_1 \times \Gamma_2$ such that $\gamma_1 = \gamma_1'$ and γ_2' is chosen nondeterministically. We then have the following lemmas:

Lemma 6 *For projections* ϖ_1, ϖ_2,

$$\begin{aligned}
[\varpi_1^{-1}];\langle\varpi_1\rangle &= skip, &\qquad \langle\varpi_1\rangle;[\varpi_1^{-1}] &\leq skip, \\
[\varpi_2^{-1}];\langle\varpi_2\rangle &= skip, &\qquad \langle\varpi_2\rangle;[\varpi_2^{-1}] &\leq skip.
\end{aligned}$$

Lemma 7 *For* $q_1, q_2 : \mathcal{P}\,\Sigma_1 \times \Sigma_2$,

$$[\varpi_1^{-1}]\ q_1 \ \times\ [\varpi_2^{-1}]\ q_2 \quad \leq \quad q_1 \wedge q_2.$$

Proof:

$$\begin{aligned}
[\varpi_1^{-1}]q_1 \ \times\ [\varpi_2^{-1}]q_2 \quad = &\qquad \{\text{Lemma } 5\} \\
&\langle\varpi_1\rangle;[\varpi_1^{-1}]\ q_1 \ \wedge\ \langle\varpi_2\rangle;[\varpi_2^{-1}]\ q_2 \\
\leq &\qquad \{\text{Lemma } 6\} \\
&q_1 \wedge q_2.
\end{aligned}$$

$\square$

Given $S_1 : \Sigma \longmapsto \Gamma_1$, the predicate transformer $S_1;[\varpi_1^{-1}] : \Sigma \longmapsto \Gamma_1 \times \Gamma_2$ is a 'lifted' version of S_1 that behaves as S_1 on the first component of the final state, and nondeterministically writes any value to the second component. Similarly for $S_2;[\varpi_2^{-1}]$. Now we have:

Theorem 29 *For* $S_1 : \Sigma \longmapsto \Gamma_1,\ \ S_2 : \Sigma \longmapsto \Gamma_2$,

$$S_1 \otimes S_2 \quad = \quad S_1;[\varpi_1^{-1}] \ \odot\ S_2;[\varpi_2^{-1}].$$

Proof: for $q : \mathcal{P}(\Gamma_1 \times \Gamma_2)$,

$$\begin{array}{ll}
& (S_1;[\varpi_1^{-1}] \ \odot \ S_2;[\varpi_2^{-1}])\ q \\
= & (\exists q_1, q_2 : \mathcal{P}(\Gamma_1 \times \Gamma_2) \mid q_1 \wedge q_2 \leq q \cdot (S_1;[\varpi_1^{-1}]\ q_1) \wedge (S_2;[\varpi_2^{-1}]\ q_2)) \\
\leq & \qquad \{\text{Lemma 7}\} \\
& (\exists q_1, q_2 : \mathcal{P}(\Gamma_1 \times \Gamma_2) \mid [\varpi_1^{-1}]q_1 \times [\varpi_2^{-1}]q_2 \leq q \cdot (S_1;[\varpi_1^{-1}]\ q_1) \wedge (S_2;[\varpi_2^{-1}]\ q_2)) \\
\leq & \qquad \{\text{take } q_1' = [\varpi_1^{-1}]q_1, \quad q_2' = [\varpi_2^{-1}]q_2\} \\
& (\exists q_1' : \mathcal{P}\,\Gamma_1;\ q_2' : \mathcal{P}\,\Gamma_2 \mid q_1' \times q_2' \leq q \cdot S_1 q_1' \ \wedge\ S_2 q_2') \\
= & (S_1 \otimes S_2)\ q.
\end{array}$$

$$\begin{array}{ll}
& (S_1 \otimes S_2)\ q \\
= & (\exists q_1 : \mathcal{P}\,\Gamma_1;\ q_2 : \mathcal{P}\,\Gamma_2 \mid q_1 \times q_2 \leq q \cdot S_1 q_1 \ \wedge\ S_2 q_2) \\
= & \qquad \{\text{Lemma 5}\} \\
& (\exists q_1 : \mathcal{P}\,\Gamma_1;\ q_2 : \mathcal{P}\,\Gamma_2 \mid \langle\varpi_1\rangle q_1 \wedge \langle\varpi_2\rangle q_2 \leq q \cdot S_1 q_1 \ \wedge\ S_2 q_2) \\
= & \qquad \{\text{Lemma 6}\} \\
& (\exists q_1 : \mathcal{P}\,\Gamma_1;\ q_2 : \mathcal{P}\,\Gamma_2 \mid \langle\varpi_1\rangle q_1 \wedge \langle\varpi_2\rangle q_2 \leq q \cdot (S_1;[\varpi_1^{-1}];\langle\varpi_1\rangle\ q_1) \wedge (S_2;[\varpi_2^{-1}];\langle\varpi_2\rangle\ q_2)) \\
\leq & \qquad \{\text{take } q_1' = \langle\varpi_1\rangle q_1, \quad q_2' = \langle\varpi_2\rangle q_2\} \\
& (\exists q_1', q_2' : \mathcal{P}(\Gamma_1 \times \Gamma_2) \mid q_1' \wedge q_2' \leq q \cdot (S_1;[\varpi_1^{-1}]\ q_1') \wedge (S_2;[\varpi_2^{-1}]\ q_2')) \\
= & (S_1;[\varpi_1^{-1}] \ \odot \ S_2;[\varpi_2^{-1}])\ q.
\end{array}$$

□

THEOREM **30** *The product operator preserves monotonicity and top and bottom homomorphism, as well as conjunctivity and disjunctivity.*

Proof: Preservation of monotonicity, top homomorphism and conjunctivity follow from Theorem 29 and the fact that they are preserved by the fusion, sequential composition, and demonic update operators. To show bottom homomorphism, assume $S_1 \perp = \perp$ and $S_2 \perp = \perp$, then

$$\begin{array}{rcll}
(S_1 \otimes S_2) \perp & = & (\exists q_1 : \mathcal{P}\,\Gamma_1;\ q_2 : \mathcal{P}\,\Gamma_2 \mid q_1 \times q_2 \leq \perp \cdot S_1 q_1 \wedge S_2 q_2) & \\
& = & (\exists q_1 : \mathcal{P}\,\Gamma_1;\ q_2 : \mathcal{P}\,\Gamma_2 \mid q_1 = \perp \vee q_2 = \perp \cdot S_1 q_1 \wedge S_2 q_2) & (*) \\
& = & (\exists q_2 : \mathcal{P}\,\Gamma_2 \cdot S_1 \perp \wedge S_2 q_2) \ \vee\ (\exists q_1 : \mathcal{P}\,\Gamma_1 \cdot S_1 q_1 \wedge S_2 \perp) & \\
& = & (\exists q_2 : \mathcal{P}\,\Gamma_2 \cdot \perp \wedge S_2 q_2) \ \vee\ (\exists q_1 : \mathcal{P}\,\Gamma_1 \cdot S_1 q_1 \wedge \perp) & \\
& = & \perp . &
\end{array}$$

The step marked * works for $q_1 \times q_2$ but not $q_1 \wedge q_2$. Proof that the product operator preserves disjunctivity is given in [21] and is repeated in the appendix here. □

Note that preservation of disjunctivity for the product operator does not follow from the fact that the fusion operator preserves disjunctivity but has to be proven separately. This is because the demonic updates $[\varpi_1^{-1}]$ and $[\varpi_2^{-1}]$ are not disjunctive. We have not been able to find a proof that the product operator preserves continuity, though neither can we find a counterexample. Naumann [21] describes a continuous product operator in which the quantification is over finitary predicates. He also shows that this corresponds to the product operator for countable state spaces when the operands are continuous, i.e., the product operator preserves continuity for countable state spaces.

Distribution of the product operator through disjunction and conjunction follows from Theorems 24 and 29. Preservation of refinement follows from Theorems 26 and 29. The product operator does not, in general, satisfy selection and separation properties, though it does satisfying the following:

THEOREM **31** *For monotonic* $S_1 : \Sigma \longmapsto \Gamma_1,\ \ S_2 : \Sigma \longmapsto \Gamma_2,\ q : \mathcal{P}\,\Gamma_1$,

$$(S_1 \otimes S_2);\langle\varpi_1\rangle\ q \quad = \quad (S_1 q \ \wedge\ S_2 \top) \ \vee\ (S_1 \top \ \wedge\ S_2 \perp).$$

Proof: Firstly, we have for $q_1 : \mathcal{P}\,\Gamma_1, q_2 : \mathcal{P}\,\Gamma_2$,

$$\begin{array}{rcl} q_1 \times q_2 \;\leq\; \langle\varpi_1\rangle q & = & (\forall \gamma_1 : \Gamma_1;\; \gamma_2 : \Gamma_2 \cdot q_1 \times q_2\ (\gamma_1, \gamma_2) \;\Rightarrow\; \langle\varpi_1\rangle q\ (\gamma_1, \gamma_2)) \\ & = & (\forall \gamma_1 : \Gamma_1;\; \gamma_2 : \Gamma_2 \cdot q_1\gamma_1 \;\wedge\; q_2\gamma_2 \;\Rightarrow\; q\ \gamma_1) \\ & = & (\forall \gamma_1 : \Gamma_1 \cdot q_1\gamma_1 \;\Rightarrow\; q\ \gamma_1) \;\vee\; (\forall \gamma_2 : \Gamma_2 \cdot \neg\, q_2\gamma_2) \\ & = & q_1 \leq q \;\vee\; q_2 = \perp . \end{array}$$

Then,

$$\begin{array}{rl} & (S_1 \otimes S_2); \langle\varpi_1\rangle\ q \\ = & (\exists q_1 : \mathcal{P}\,\Gamma_1;\; q_2 : \mathcal{P}\,\Gamma_2 \,|\, q_1 \times q_2 \leq \langle\varpi_1\rangle q \cdot S_1 q_1 \;\wedge\; S_2 q_2) \\ = & (\exists q_1 : \mathcal{P}\,\Gamma_1;\; q_2 : \mathcal{P}\,\Gamma_2 \,|\, q_1 \leq q \cdot S_1 q_1 \;\wedge\; S_2 q_2) \;\vee\; (\exists q_1 : \mathcal{P}\,\Gamma_1;\; q_2 : \mathcal{P}\,\Gamma_2 \,|\, q_2 = \perp \cdot S_1 q_1 \;\wedge\; S_2 q_2) \\ = & \qquad \{S_2 \text{ monotonic}\} \\ & (\exists q_1 : \mathcal{P}\,\Gamma_1 \,|\, q_1 \leq q \cdot S_1 q_1) \;\wedge\; S_2 \top \quad \vee \quad (\exists q_1 : \mathcal{P}\,\Gamma_1 \cdot S_1 q_1) \;\wedge\; S_2 \perp \\ = & \qquad \{S_1 \text{ monotonic}\} \\ & S_1 q \;\wedge\; S_2 \top \quad \vee \quad S_1 \top \;\wedge\; S_2 \perp . \end{array}$$

□

If S_2 is $\perp$-homomorphic, we get

$$(S_1 \otimes S_2); \langle\varpi_1\rangle \quad \leq \quad S_1 .$$

If S_2 is both $\perp$- and $\top$-homomorphic, this becomes an equality. A similar result holds for $(S_1 \otimes S_2); \langle\varpi_2\rangle$.

The separation property does not hold for the product operator, though we do have the following inequality for conjunctive S:

THEOREM **32** *For conjunctive* $S : \Sigma \longmapsto \Gamma_1 \times \Gamma_2$,

$$S; \langle\varpi_1\rangle \;\otimes\; S; \langle\varpi_2\rangle \quad \leq \quad S.$$

Proof: For $q : \mathcal{P}\,(\Gamma_1 \times \Gamma_2)$,

$$\begin{array}{rcl} (S; \langle\varpi_1\rangle \;\otimes\; S; \langle\varpi_2\rangle)\ q & = & (\exists q_1 : \mathcal{P}\,\Gamma_1;\; q_2 : \mathcal{P}\,\Gamma_2 \,|\, q_1 \times q_2 \leq q \cdot S; \langle\varpi_1\rangle\ q_1 \;\wedge\; S; \langle\varpi_2\rangle\ q_2) \\ & = & \qquad \{S \text{ conjunctive}\} \\ & & (\exists q_1 : \mathcal{P}\,\Gamma_1;\; q_2 : \mathcal{P}\,\Gamma_2 \,|\, q_1 \times q_2 \leq q \cdot S(\langle\varpi_1\rangle\ q_1 \;\wedge\; \langle\varpi_2\rangle\ q_2)) \\ & = & \qquad \{\text{Lemma } 5\} \\ & & (\exists q_1 : \mathcal{P}\,\Gamma_1;\; q_2 : \mathcal{P}\,\Gamma_2 \,|\, q_1 \times q_2 \leq q \cdot S(q_1 \times q_2)) \\ & \leq & \qquad \{S \text{ monotonic}\} \\ & & S\ q. \end{array}$$

□

The reverse inequality does not hold for a command that chooses nondeterministically from some non-rectangular subset of $\Gamma_1 \times \Gamma_2$. For example, take

$$S \quad = \quad [\lambda\sigma \cdot \lambda\gamma_1, \gamma_2 \cdot \gamma_1 = \gamma_2].$$

We have that $S; \langle\varpi_1\rangle = havoc$, and $S; \langle\varpi_1\rangle \otimes S; \langle\varpi_1\rangle = havoc$.

The following refinement theorem follows directly from Theorem 29 and Theorem 27:

THEOREM **33**

$$\begin{array}{rcl} \{S_2 \top\}; S_1; [\varpi_1^{-1}] & \leq & S_1 \otimes S_2 \\ \{S_1 \top\}; S_2; [\varpi_2^{-1}] & \leq & S_1 \otimes S_2 . \end{array}$$

In order to consider the product of deterministic updates and specification statements, we require product operators for functions and relations:

DEFINITION **9** *For functions* $f_1 : \Sigma \rightarrow \Gamma_1$, $f_2 : \Sigma \rightarrow \Gamma_2$, $f_1 \otimes f_2$ *is a function of type* $\Sigma \rightarrow (\Gamma_1 \times \Gamma_2)$, *where for* $\sigma : \Sigma$,

$$f_1 \otimes f_2 \;\; \sigma \;\; \widehat{=} \;\; (f_1 \; \sigma, f_2 \; \sigma).$$

DEFINITION **10** *For relations* $P_1 : \Sigma \leftrightarrow \Gamma_1$, $P_2 : \Sigma \leftrightarrow \Gamma_2$, $P_1 \otimes P_2$ *is a relation of type* $\Sigma \leftrightarrow (\Gamma_1 \times \Gamma_2)$, *where for* $\sigma : \Sigma$, $\gamma_1 : \Gamma_1$, $\gamma_2 : \Gamma_2$,

$$P_1 \otimes P_2 \; \sigma \; (\gamma_1, \gamma_2) \;\; \widehat{=} \;\; P_1 \; \sigma \; \gamma_1 \;\; \wedge \;\; P_2 \; \sigma \; \gamma_2.$$

The product operator then combines with specification statements and deterministic updates in the following manner:

THEOREM **34** *For predicates* $p_1, p_2 : \mathcal{P}\,\Sigma$, *and relations* $P_1 : \Sigma \leftrightarrow \Gamma_1$, $P_2 : \Sigma \leftrightarrow \Gamma_2$,

$$\{p_1\};[P_1] \;\otimes\; \{p_2\};[P_2] \;\;=\;\; \{p_1 \wedge p_2\};[P_1 \otimes P_2].$$

Proof: It is easy to show that

$$P_1 \times P_2 \;\;=\;\; P_1;\varpi_1^{-1} \;\;\wedge\;\; P_2;\varpi_2^{-1}.$$

Result then follows from Theorem 25 (fusion of specification statements). □

THEOREM **35** *For functions* $f_1 : \Sigma \rightarrow \Gamma_1$, $f_2 : \Sigma \rightarrow \Gamma_2$,

$$\langle f_1 \rangle \otimes \langle f_2 \rangle \;\;=\;\; \langle f_1 \otimes f_2 \rangle.$$

Proof: Follows easily from Theorem 34 since for any f, $\langle f \rangle = \{\top\};[\tilde{f}]$. □

4.3 Properties of the Derived Product Operator

Theorem 29 and Definition 7 together give us:

THEOREM **36** *For* $S_1 : \Sigma_1 \longmapsto \Gamma_1$, $\;S_2 : \Sigma_2 \longmapsto \Gamma_2$,

$$S_1 \times S_2 \;\;=\;\; \langle \pi_1 \rangle;S_1;[\varpi_1^{-1}] \;\odot\; \langle \pi_2 \rangle;S_2;[\varpi_2^{-1}].$$

Here, the command $\langle \pi_1 \rangle$ starts in a state $(\sigma_1, \sigma_2) : \Sigma_1 \times \Sigma_2$ and ends simply in the state σ_1. Given $S_1 : \Sigma_1 \longmapsto \Gamma_1$, the predicate transformer $\langle \pi_1 \rangle;S_1;[\varpi_1^{-1}] : \Sigma_1 \times \Sigma_2 \longmapsto \Gamma_1 \times \Gamma_2$ is a 'lifted' version of S_1 that behaves as S_1 on the first components of the before and after state, and behaves nondeterministically on the second components. Similarly for $\langle \pi_2 \rangle;S_1;[\varpi_1^{-1}]$.

Preservation of monotonicity and homomorphic properties follows directly from Theorems 36 and 30:

THEOREM **37** *The derived product operator (Definition 7) preserves monotonicity, bottom and top homomorphism, and conjunctivity and disjunctivity.*

The derived product operator may also be characterised in disjunctive form:

THEOREM **38** *For* $S_1 : \Sigma_1 \longmapsto \Gamma_1$, $\;S_2 : \Sigma_2 \longmapsto \Gamma_2$, $\;q : \mathcal{P}\,(\Gamma_1 \times \Gamma_2)$:

$$(S_1 \times S_2)\; q \;\;=\;\; (\exists q_1 : \mathcal{P}\,\Gamma_1;\; q_2 : \mathcal{P}\,\Gamma_2 \mid q_1 \times q_2 \leq q \cdot S_1 q_1 \times S_2 q_2).$$

Proof:

$$\begin{aligned}
(S_1 \times S_2)\; q \;\;&=\;\; (\langle \pi_1 \rangle;S_1 \;\otimes\; \langle \pi_2 \rangle;S_2)\; q \\
&=\;\; (\exists q_1 : \mathcal{P}\,\Gamma_1;\; q_2 : \mathcal{P}\,\Gamma_2 \mid q_1 \times q_2 \leq q \cdot (\langle \pi_1 \rangle;S_1\; q_1) \wedge (\langle \pi_2 \rangle;S_2\; q_2)) \\
&=\;\; \qquad \{\text{Lemma 5}\} \\
&\qquad (\exists q_1 : \mathcal{P}\,\Gamma_1;\; q_2 : \mathcal{P}\,\Gamma_2 \mid q_1 \times q_2 \leq q \cdot S_1 q_1 \times S_2 q_2).
\end{aligned}$$

□

The following results, showing how the derived product operator combines with specification statements and deterministic updates, follow directly from Theorems 34 and 35:

DEFINITION **11** *For functions $f_1 : \Sigma_1 \rightarrow \Gamma_1$, $f_2 : \Sigma_2 \rightarrow \Gamma_2$, $f_1 \times f_2$ is a function of type $(\Sigma_1 \times \Sigma_2) \rightarrow (\Gamma_1 \times \Gamma_2)$, where for $\sigma_1 : \Sigma_1$, $\sigma_2 : \Sigma_2$,*

$$f_1 \times f_2\ (\sigma_1, \sigma_2) \ \ \widehat{=}\ \ (f_1\sigma_1, f_2\sigma_2).$$

DEFINITION **12** *For relations $P_1 : \Sigma_1 \leftrightarrow \Gamma_1$, $P_2 : \Sigma_2 \leftrightarrow \Gamma_2$, $P_1 \times P_2$ is a relation of type $(\Sigma_1 \times \Sigma_2) \leftrightarrow (\Gamma_1 \times \Gamma_2)$, where for $\sigma_1 : \Sigma_1$, $\sigma_2 : \Sigma_2$, $\gamma_1 : \Gamma_1$, $\gamma_2 : \Gamma_2$,*

$$P_1 \times P_2\ (\sigma_1, \sigma_2)\ (\gamma_1, \gamma_2) \ \ \widehat{=}\ \ (P_1\ \sigma_1\ \gamma_1)\ \ \wedge\ \ (P_2\ \sigma_2\ \gamma_2).$$

THEOREM **39** *For functions $f_1 : \Sigma_1 \rightarrow \Gamma_1$, $f_2 : \Sigma_2 \rightarrow \Gamma_2$,*

$$\langle f_1 \rangle \times \langle f_2 \rangle \ \ = \ \ \langle f_1 \times f_2 \rangle.$$

THEOREM **40** *For predicates $p_1 : \mathcal{P}\,\Sigma_1$, $p_2 : \mathcal{P}\,\Sigma_2$, and relations $P_1 : \Sigma_1 \leftrightarrow \Gamma_1$, $P_2 : \Sigma_2 \leftrightarrow \Gamma_2$,*

$$\{p_1\}; [P_1] \ \times\ \{p_2\}; [P_2] \ \ = \ \ \{p_1 \times p_2\}; [P_1 \times P_2].$$

Distribution of the derived product operator through disjunction and conjunction follows from Theorems 24 and 29. Preservation of refinement follows from Theorems 26 and 36.

Like the product operator, the derived product operator only satisfies the selection inequality for $\perp$-homomorphic predicate transformers:

$$[\pi_1^{-1}]; (S_1 \times S_2); \langle \varpi_1 \rangle \ \leq\ S_1 \qquad\qquad [\pi_2^{-1}]; (S_1 \times S_2); \langle \varpi_2 \rangle \ \leq\ S_2.$$

The inequalities become equalities if S_1 and S_2 are also $\top$-homomorphic.

The derived product operator does not satisfy a separation property. For example, the command

$$\langle \lambda(x, y) : \Sigma \times \Sigma \cdot (y, x) \rangle : \Delta(\Sigma \times \Sigma)$$

that simply swaps the values of two state components, cannot be separated into two commands of type $\Delta(\Sigma)$. Recall that the same was true of the derived summation operator.

Unlike the derived summation operator, the derived product operator does not distribute through recursion. For example, we have that

$$\begin{aligned} \textbf{do}\ \mathit{skip}\ \textbf{od}\ \times\ \textbf{do}\ \mathit{magic}\ \textbf{od} \ &= \ \mathit{abort} \times \mathit{skip} \\ &= \ \mathit{abort}, \end{aligned}$$

while,

$$\begin{aligned} \textbf{do}\ \mathit{skip} \times \mathit{magic}\ \textbf{od} \ &= \ \textbf{do}\ \mathit{magic}\ \textbf{od} \\ &= \ \mathit{skip}. \end{aligned}$$

The product operators distribute through sequential composition as follows:

THEOREM **41** *For $R_1 : \Sigma \longmapsto \Gamma_1$, $R_2 : \Sigma \longmapsto \Gamma_2$, $S_1 : \Sigma_1 \longmapsto \Gamma_1$, $S_2 : \Sigma_2 \longmapsto \Gamma_2$, $T_1 : \Gamma_1 \longmapsto \Lambda_1$, $T_2 : \Gamma_2 \longmapsto \Lambda_2$,*

$$\begin{aligned} (R_1; T_1) \otimes (R_2; T_2) \ &\leq\ (R_1 \otimes R_2); (T_1 \times T_2) \\ (S_1; T_1) \times (S_2; T_2) \ &\leq\ (S_1 \times S_2); (T_1 \times T_2). \end{aligned}$$

In the case that T_1 and T_2 are universally conjunctive, then the inequalities become equalities.

Proof: See [5].

This result allows us to show how the derived product operator preserve data-refinement. First we note the following from [26]: for predicate transformer S, the *right adjoint* of S, denoted S^r, satisfies

$$S; S^r \ \leq\ \mathit{skip} \qquad\qquad \mathit{skip} \ \leq\ S^r; S.$$

S has a right adjoint if and only if S is universally disjunctive. In this case, S^r is universally conjunctive. For example, for relation P, $\{P\}^r = [P^{-1}]$. In the case that abstraction command α is universally disjunctive, we have:

$$\alpha; S \ \leq\ S'; \alpha \quad\ \Leftrightarrow\ \quad S; \alpha^r \ \leq\ \alpha^r; S'.$$

Now, we have the following theorem for disjunctive abstraction commands:

THEOREM **42** *Let* $S_1 : \Delta(\Sigma_1)$, $S_1' : \Delta(\Sigma_1')$, $S_2 : \Delta(\Sigma_2)$, $S_2' : \Delta(\Sigma_2')$, *be predicate transformers. Let* $\alpha_1 : \Sigma_1' \longmapsto \Sigma_1$, $\alpha_2 : \Sigma_2' \longmapsto \Sigma_2$ *be abstraction commands. If* α_1 *and* α_2 *are universally disjunctive, then*

$$\begin{array}{l} S_1; \alpha_1^r \leq \alpha_1^r; S_1' \quad \wedge \quad S_1; \alpha_1^r \leq \alpha_1^r; S_1' \ \Rightarrow \\ \qquad (S_1 \times S_2); (\alpha_1^r \times \alpha_2^r) \ \leq \ (\alpha_1^r \times \alpha_2^r); (S_1' \times S_2'). \end{array}$$

Proof:

$$\begin{array}{rcl} (S_1 \times S_2); (\alpha_1^r \times \alpha_2^r) & = & \qquad \{\text{Theorem 41, } \alpha_1^r \text{ and } \alpha_2^r \text{ universally conjunctive}\} \\ & & (S_1; \alpha_1^r) \times (S_2; \alpha_2^r) \\ & \leq & \qquad \{\times \text{ preserves refinement}\} \\ & & (\alpha_1^r; S_1') \times (\alpha_2^r; S_2') \\ & \leq & \qquad \{\text{Theorem 41}\} \\ & & (\alpha_1^r \times \alpha_2^r); (S_1' \times S_2'). \end{array}$$

□

Note that the case where α is universally disjunctive corresponds to downward (or forward) data refinement [26].

5 Categorical Properties

In this section, we briefly consider some categorical properties of the summation and product operators. A *category* $\mathcal{C}$ is a pair (O, A) of objects O and arrows A. Associated with each arrow f in A is a source object, $source(f) \in O$, and a target object, $target(f) \in O$. For $a, b \in O$, we write $f : a \rightarrow b$ if f is an arrow of A with source a and target b. Associated with each object $a \in O$, there is an identity arrow $i_a : a \rightarrow a$, and with each pair of arrows $f : a \rightarrow b$ and $g : b \rightarrow c$ there is the composition of f and g, denoted $f; g : a \rightarrow c$. For objects a, b, c, d, and arrows $f : a \rightarrow b$, $g : b \rightarrow c$ and $h : c \rightarrow d$, the following holds:

$$i_a; f \ = \ f \ = \ f; i_b, \tag{8}$$

$$f; (g; h) \ = \ (f; g); h. \tag{9}$$

Predicate transformers form a category: the category $PTran$ has state spaces as objects and predicate transformers as arrows. If Σ, Γ are sets, then each predicate transformer $S : \Sigma \longmapsto \Gamma$ is an arrow with source Σ and target Γ. For composition of arrows we use sequential composition, and as the identity arrow of object Σ we use $skip_\Sigma$. It is easy to see that (8) and (9) above are satisfied.

Let a and b be objects in a category $\mathcal{C}$. Then a (categorical) *product* of a and b is an object $a \times b$, together with a pair of projection arrows $proj_1 : a \times b \rightarrow a$ and $proj_2 : a \times b \rightarrow b$, such that for all objects c and arrows $f : c \rightarrow a$ and $g : c \rightarrow b$, there is a unique arrow $f \otimes g$[2] satisfying

$$(f \otimes g); proj_1 \ = \ f \quad \wedge \quad (f \otimes g); proj_2 \ = \ g. \tag{10}$$

A *coproduct* is the dual of a product: a (categorical) coproduct of a and b is an object $a + b$, together with a pair of injection arrows $inj_1 : a \rightarrow a + b$ and $inj_2 : b \rightarrow a + b$, such that for all objects c and arrows $f : a \rightarrow c$ and $g : b \rightarrow c$, there is a unique arrow $f \oplus g$ satisfying

$$inj_1; (f \oplus g) \ = \ f \quad \wedge \quad inj_2; (f \oplus g) \ = \ g. \tag{11}$$

It is easy to see that the summation operator for predicate transformers yields a categorical coproduct: Let ϕ_1 and ϕ_2 be the injections for $\Sigma_1 + \Sigma_2$. By using the updates $\langle \phi_1 \rangle$ and $\langle \phi_2 \rangle$ as injection arrows, we have, by the selection property of the summation operator (Theorem 3), that for predicate transformers $S_1 : \Gamma \longmapsto \Sigma_1$ and $S_2 : \Gamma \longmapsto \Sigma_2$, the summation $S_1 \oplus S_1$ satisfies (11). Furthermore, the separation property of the summation operator shows that $S_1 \oplus S_1$ is the unique predicate transformer satisfying

[2] In category theory literature, this is usually written $\langle f, g \rangle$ while the coproduct arrow is written $[f, g]$.

(11): assume X satisfies (11), i.e., $\langle\phi_1\rangle; X = S_1$ and $\langle\phi_2\rangle; X = S_2$, then

$$\begin{array}{rl} X & = \quad \{\text{Separation property}\} \\ & \langle\phi_1\rangle; X \;\oplus\; \langle\phi_2\rangle; X \\ = & \quad \{\text{assumption}\} \\ & S_1 \oplus S_2 . \end{array}$$

Thus, for objects Σ_1 and Σ_2, the summation $\Sigma_1 + \Sigma_2$, together with the injections $\langle\phi_1\rangle$ and $\langle\phi_2\rangle$, is a coproduct in $PTran$.

The product operator for predicate transformers does not yield a categorical product since it does not satisfy a selection property (c.f. Theorem 31). However, Theorems 31 and 32 tell us that the product operator does satisfy weaker selection and separation inequalities on predicate transformers that are $\perp$-homomorphic and conjunctive:

$$(S_1 \otimes S_2); \langle\varpi_1\rangle \;\leq\; S_1 \qquad (S_1 \otimes S_2); \langle\varpi_2\rangle \;\leq\; S_2$$
$$S; \langle\varpi_1\rangle \;\otimes\; S; \langle\varpi_2\rangle \;\leq\; S.$$

This is called a *weak* product in [15]. Let $PTran_{\perp,\wedge}$ be the category of $\perp$-homomorphic and conjunctive predicate transformers, i.e., Dijkstra's original command language. For objects S_1 and S_2 of $PTran_{\perp,\wedge}$, $S_1 \otimes S_2$ is also an object of $PTran_{\perp,\wedge}$. Thus the product operator for predicate transformers yields a weak categorical product in $PTran_{\perp,\wedge}$ (a result that is known from [15]).

6 Applications

6.1 Simultaneous Assignment

As mentioned in Section 2, variables may be modelled as cartesian components of the state space. We take the view that the predicate transformer $S_1 : \Sigma_1 \times \Sigma_2 \longmapsto \Sigma_1$ models a statement that reads from program variables x and y, and assigns a value to program variable x, while $S_2 : \Sigma_1 \times \Sigma_2 \longmapsto \Sigma_2$ reads from the same variables and assigns a value to y. The product $S_1 \otimes S_2$ has type $\Delta(\Sigma_1 \times \Sigma_2)$ and models simultaneous assignment to x and y. For example, we have

$$\begin{array}{rcl} x := f(x,y) \;\otimes\; y := g(x,y) & = & \langle \lambda x, y \cdot f(x,y)\rangle \;\otimes\; \langle \lambda x, y \cdot g(x,y)\rangle \\ & = & \quad \{\text{Theorem 35}\} \\ & & \langle \lambda x, y \cdot f(x,y), g(x,y)\rangle \\ & = & x, y := f(x,y), g(x,y). \end{array}$$

If S_1 and S_2 read from distinct state components, i.e., S_1 has type $\Delta(\Sigma_1)$ and S_2 has type $\Delta(\Sigma_2)$, then their simultaneous execution is modelled by the derived product $S_1 \times S_2$. With both product operators, we know that if either assignment is nonterminating, then the combined assignment is nonterminating, e.g., $S_1 \otimes \mathit{abort} \;=\; \mathit{abort}$.

The *action system* formalism of Back & Kurki-Suonio [2] uses predicate transformers to model parallel programs. An action system consists of an initialisation predicate transformer and a set of action predicate transformers. Execution of an action system proceeds by firstly executing the initialisation, then, repeatedly, executing an enabled action (an action A is enabled when $gd\ A$ holds). Two action systems are composed in parallel by composing their initialisations such that they are executed simultaneously and forming the union of their actions. Conventionally, the initialisations are demonic updates $[I_1]$, $[I_2]$, and their composition is simply $[I_1 \wedge I_2]$. The product operator provides a way of composing more general initialisations achieving the same effect.

In [7], the actions of an action system are given labels and a correspondence with Hoare's CSP [13] is established. A version of parallel composition of action systems is described in which commonly labelled actions from the respective action systems are composed such that they are executed simultaneously. In [6], this composition is defined by the properties that it should satisfy. One such property is that the composition should only be enabled when both actions are enabled. The product operator almost satisfies all the required properties: it fails if one of the actions can abort. However, by guarding the product of the actions as follows, we get an operator that satisfies the required properties:

$$A_1 \| A_2 \;\;\widehat{=}\;\; [(gd\ A_1) \times (gd\ A_2)]; A_1 \times A_2 .$$

6.2 Superposition

Consider the following two predicate transformers:

$$S_1 : \Delta(\Sigma_1), \qquad S_2 : \Sigma_1 \times \Sigma_2 \longmapsto \Sigma_2.$$

S_1 only operates on Σ_1, while S_2 reads from $\Sigma_1 \times \Sigma_2$ and writes to Σ_2. Superposition of S_2 on to S_1 allows S_2 to add extra program variables which it writes to, as well as allowing S_2 to read the program variables of S_1. Such an operation may be defined as a special case of the product operator:

DEFINITION **13** $\quad S_1 \,\mathbf{sup}\, S_2 \;\widehat{=}\; \langle \pi_1 \rangle; S_1 \;\otimes\; S_2.$

Here, $\langle \pi_1 \rangle$ reads from $\Sigma_1 \times \Sigma_2$ though it discards the Σ_2 component. For example, we have:

$$x := f(x) \quad \mathbf{sup} \quad y := g(x,y) \quad = \quad x, y := f(x), g(x,y).$$

Superposition satisfies the following refinement rule:

THEOREM **43** $\quad \{S_2 \top\}; \langle \pi_1 \rangle; S_1 \;\leq\; (S_1 \,\mathbf{sup}\, S_2); \langle \pi_1 \rangle.$

Proof:

$$\begin{array}{rl}
\{S_2 \top\}; \langle \pi_1 \rangle; S_1 = & \quad \{\ [\pi_1^{-1}]; \langle \pi_1 \rangle = \mathit{skip} \text{ (Lemma 6) } \} \\
& \{S_2 \top\}; (\langle \pi_1 \rangle; S_1); [\pi_1^{-1}]; \langle \pi_1 \rangle \\
\leq & \quad \{\text{Theorem 33}\} \\
& (\langle \pi_1 \rangle; S_1 \;\otimes\; S_2); \langle \pi_1 \rangle \\
= & (S_1 \,\mathbf{sup}\, S_2); \langle \pi_1 \rangle.
\end{array}$$

□

In the case that $S_2 \top = \top$, then this theorem shows that superposition is a form of data refinement with $\langle \pi_1 \rangle$ as the abstraction statement.

Superposition refinement of action systems is described in [3], where superposition on individual actions is described in terms of sequential composition. Our superposition operator could be used instead.

6.3 Modification

Consider the following two predicate transformers:

$$S_1 : \Delta(\Sigma_1 \times \Sigma_2), \qquad S_2 : \Sigma_1 \times \Sigma_2 \longmapsto \Sigma_2.$$

S_1 operates on $\Sigma_1 \times \Sigma_2$, while S_2 reads from $\Sigma_1 \times \Sigma_2$ and writes to Σ_2. For example, we could have

$$S_1 \;=\; x, y := f(x,y), g(x,y), \qquad S_2 \;=\; y := h(x,y).$$

We wish to modify S_1 with S_2 so that S_1 writes to Σ_1 as before, but the value written to Σ_2 is determined instead by S_2. Again, such an operation may be defined as a special case of the product operator:

DEFINITION **14** $\quad S_1 \,\mathbf{mod}\, S_2 \;\widehat{=}\; S_1; \langle \pi_1 \rangle \;\otimes\; S_2.$

Here, $\langle \pi_1 \rangle$ discards the Σ_2 component written to by S_1 preserving only the Σ_1 component. For example,

$$x, y := f(x,y), g(x,y) \quad \mathbf{mod} \quad y := h(x,y) \quad = \quad x, y := f(x,y), h(x,y).$$

Modification preserves refinement:

THEOREM **44** $\quad S_1 \leq S_1' \;\wedge\; S_2 \leq S_2' \;\Rightarrow\; (S_1 \,\mathbf{mod}\, S_2) \leq (S_1' \,\mathbf{mod}\, S_2').$

6.4 Rearranging and Extending State Components

Sometimes when combining statements, their state components may not match in the required way. For example, the statements $S_1 : \Delta(\Sigma_1 \times \Sigma_2 \times \Sigma_2)$ and $S_2 : \Delta(\Sigma_2 \times \Sigma_3 \times \Sigma_1)$ may not be combined directly as $S_1; S_2$, (even if Σ_1, Σ_2, and Σ_3 are the same sets, constructing $S_1; S_2$ directly will result in the state components being mismatched). Instead, the state components may be rearranged using an update statement before combining the statements:

$$S_1; \langle\delta\rangle; S_2; \langle\delta^{-1}\rangle,$$

where $\delta(\sigma_1, \sigma_2, \sigma_3) = (\sigma_2, \sigma_3, \sigma_1)$.

The same technique can be used to match statements modulo associativity. For example, although

$$(S_1 \times S_2) \times S_3 \quad \neq \quad S_1 \times (S_2 \times S_3),$$

we do have (let $\kappa((\sigma_1, \sigma_2), \sigma_2) \quad = \quad (\sigma_1, (\sigma_2, \sigma_2))$):

$$(S_1 \times S_2) \times S_3 \quad = \quad \langle\kappa\rangle; (S_1 \times (S_2 \times S_3)); \langle\kappa^{-1}\rangle.$$

Sometimes the state components of a statement need to be extended in order to combine it with another statement. For example, suppose we wish to combine $S_1 : \Delta(\Sigma_1 \times \Sigma_2)$ and $S_2 : \Delta(\Sigma_2)$. Here, S_2 is independent of Σ_1, so when combining it with S_1 we would most likely expect it to leave the Σ_1 component unchanged. To achieve this, we first combine S_2 with *skip*:

$$S_1; (skip_{\Sigma_1} \times S_2).$$

Extension of the state space is required, for example, when composing action systems: the state space of all actions must be extended to the state space of the combined action system.

In [4], the block construct is used to localise state components: if S has type $\Delta(\Sigma_1 \times \Sigma_2)$, then $(\mathbf{block}\ S)$ has type $\Delta(\Sigma_1)$, where

$$(\mathbf{block}\ S)\ q_1\ \sigma_1 \quad = \quad (\forall \sigma_2 \cdot S\ (\lambda(\sigma_1', \sigma_2') \cdot q_1 \sigma_1')\ (\sigma_1, \sigma_2)).$$

The same effect can be achieved by constructing $[\pi_1^{-1}]; S; \langle\pi_1\rangle$. It is easy to show that

$$(\mathbf{block}\ S) \quad = \quad [\pi_1^{-1}]; S; \langle\pi_1\rangle.$$

6.5 Combining Specifications

A program specification takes the form $\{p\}; [P]$ where p is a state predicate and P is a state relation. The fusion operator can be used as a (refinement preserving) way of combining program specifications. By Theorem 27 $\{p\}; [P] \ \odot\ \{q\}; [Q]$ satisfies:

$$\begin{aligned} \{p \wedge q\}; [P] \quad &\leq \quad \{p\}; [P] \ \odot\ \{q\}; [Q] \\ \{p \wedge q\}; [Q] \quad &\leq \quad \{p\}; [P] \ \odot\ \{q\}; [Q], \end{aligned}$$

and is the least-refined such specification. Theorem 25 provides an easy way of calculating the combination of two specifications:

$$\{p\}; [P] \ \odot\ \{q\}; [Q] \quad = \quad \{p \wedge q\}; [P \wedge Q].$$

If P and Q are contradictory, i.e., $P \wedge Q = \perp$, then their fusion is miraculous. Ward [27] has defined a similar combinator specifically for specification statements (which is defined by combining relations) though the combination behaves as *abort* when P and Q are contradictory. In Ward's case, a contradictory combination may be refined by any statement, whereas in our case, it is unimplementable.

6.6 Shared Variable Assignment

The fusion operator may be viewed as a way of combining statements such as that they both assign a value that they agree on to the same state component. In the case that two statements have only partially overlapping state components, then they need to be extended firstly with the *havoc* statement, before being combined using the fusion operator. For example, $S_1 : \Delta(\Sigma_1 \times \Sigma_2)$ and $S_2 : \Delta(\Sigma_2 \times \Sigma_3)$ may be combined as follows:

$$(S_1 \times havoc_{\Sigma_3}) \odot (havoc_{\Sigma_1} \times S_2).$$

Since *havoc* acts as a unit for the fusion operator, the effect of this statement on the Σ_1 component will be determined purely by S_1, and the effect on the Σ_3 component will be determined by Σ_3. The effect on the Σ_2 component will be the intersection of the nondeterministic possibilities allowed by S_1 and S_2. If that intersection is empty, then, provided S_1 and S_2 terminate, the combination behaves as magic.

Note that the above operands of the fusion operator, do not match exactly: the first is of type $\Delta((\Sigma_1 \times \Sigma_2) \times \Sigma_3)$, while the second is of type $\Delta(\Sigma_1 \times (\Sigma_2 \times \Sigma_3))$. However, these can easily be made to match using the function $(\lambda((\sigma_1, \sigma_2), \sigma_3) \cdot (\sigma_1, (\sigma_2, \sigma_3)))$ as described in Subsection 6.4.

6.7 Late Binding of Procedures

We model a procedure by simply associating a statement with a procedure name. Calling a procedure then involves executing the statement associated with the procedure name. The following example shows the concrete syntax used to describe procedures:

$$\begin{array}{l} \textbf{proc } \mathit{Increment} \\ \quad \textbf{var } x : \mathit{Int} \\ \quad x := x + 1 \end{array}$$

The concept of *late binding* of procedures is important in object-oriented programming. Late binding means that the effect of executing a procedure depends on the value of the state on which it is to be executed. For example, Utting & Robinson [25] model late binding in the refinement calculus by regarding a procedure as being a function from values to statements:

$$proc \; : \; \Sigma \rightarrow \Delta(\Sigma).$$

The effect of executing procedure *proc* in state σ is then determined by the statement $(proc\ \sigma)$. This is sometimes referred to as *instance-centered* late binding [10].[3]

A simpler notion of late binding uses types rather than values to determine which statement is selected when a procedure is called (so-called *class-centered* late binding [10]). Here we show how the summation operator may be used to model this form of late binding.

Firstly, the following operator lifts predicate transformers in either base type of a summation to the summation type:

Definition **15** *Let* $S_1 : \Sigma_1 \longmapsto \Gamma_1$ *and* $S_2 : \Sigma_2 \longmapsto \Gamma_2$ *be predicate transformers. Then,*

$$S_1^+ \mathrel{\widehat{=}} S_1 + abort \qquad\qquad S_2^+ \mathrel{\widehat{=}} abort + S_2.$$

Now, given predicate transformers

$$\begin{array}{ll} S_1 : \Sigma_1 \longmapsto \Gamma_1 \qquad & S_2 : \Sigma_2 \longmapsto \Gamma_2 \\ T_1 : \Gamma_1 \longmapsto \Lambda_1 \qquad & T_2 : \Gamma_2 \longmapsto \Lambda_2, \end{array}$$

we have the following result:

Theorem **45**

$$\begin{array}{rcl} S_1^+;(T_1+T_2) & = & (S_1;T_1)^+ \\ S_2^+;(T_1+T_2) & = & (S_2;T_2)^+. \end{array}$$

[3] Sometimes the terms "late binding" and "dynamic binding" are used interchangeably in the literature. We prefer to reserve the term "dynamic binding" for the case where the procedure associated with an instance/class may change during execution. Such an effect may be achieved, for example, by using stored procedures as provided in Oberon [20] where procedures are themselves values. A predicate transformer model of stored procedures may be found in [22].

That is, the effect of $(T_1 + T_2)$ depends on whether S_1 or S_2 is executed beforehand.
Proof:

$$\begin{array}{rcl} S_1^+;(T_1+T_2) & = & (S_1+abort);(T_1+T_2) \\ & = & \quad \{\text{Distribution of + and ;}\} \\ & & (S_1;T_1)+(abort;T_2) \\ & = & (S_1;T_1)+abort \\ & = & (S_1;T_1)^+ \end{array}$$

Similarly for $S_2^+;(T_1+T_2)$. □

Consider the following two procedures:

proc $Invert_1$	**proc** $Invert_2$
var $x : Int$	**var** $x :$ **Bool**
$x := -x$	$x := \neg\, x$

Let $Invert \mathrel{\widehat{=}} Invert_1 + Invert_2$. Then

$$\begin{array}{rcl} (x:=10)^+;Invert & = & (x:=10)^+;(Invert_1+Invert_2) \\ & = & (x:=10;Invert_1)^+ \\ & = & (x:=-10)^+, \end{array}$$

while

$$\begin{array}{rcl} (x:=true)^+;Invert & = & (x:=true)^+;(Invert_1+Invert_2) \\ & = & (x:=true;Invert_2)^+ \\ & = & (x:=false)^+. \end{array}$$

Thus the effect of $Invert$ depends on whether an integer or boolean value has been assigned to x beforehand.

The following refinement rule for summation follows trivially from Theorem 14:

Theorem **46** *For any* S_2, $\quad S_1^+ \le S_1 + S_2.$

6.8 Inheritance

Consider the following object-oriented definition:

object R
inherits S
extended by E
modified by M

Here, R inherits the behaviour of S, extends the behaviour of S by E and modifies the behaviour of S by M.

Suppose S, E and M have the following types:

$$\begin{array}{l} S : \Delta(\Sigma_1 \times \Sigma_2) \\ E : \Sigma_1 \times \Sigma_2 \times \Sigma_3 \longmapsto \Sigma_3 \\ M : \Sigma_1 \times \Sigma_2 \times \Sigma_3 \longmapsto \Sigma_2. \end{array}$$

That is, E introduces a new component of type Σ_3, while M modifies how S behaves on the Σ_2 component. One way to define R would be as follows:

$$R \mathrel{\widehat{=}} (S \,\mathbf{sup}\, E) \,\mathbf{mod}\, M.$$

However, in object-oriented programming, as well as performing the extended and modified behaviour on the extended type of R, we would expect R to behave as S on values of the type of S. But some of the behaviour of S has been discarded in $(S \,\mathbf{sup}\, E) \,\mathbf{mod}\, M$. So instead we define R as follows:

$$R \mathrel{\widehat{=}} S + ((S \,\mathbf{sup}\, E) \,\mathbf{mod}\, M).$$

Now, R uses late binding when choosing whether to perform the inherited behaviour or the extended/modified behaviour.

From Theorem 46, we have that

$$S^{+} \quad \leq \quad R.$$

Also, this form of inheritance preserves refinement:

THEOREM **47**

$$\begin{array}{l} S \leq S \;\wedge\; M \leq M' \;\wedge\; E \leq E' \;\Rightarrow \\ S + ((S\,\mathbf{sup}\,E)\,\mathbf{mod}\,M) \;\leq\; S' + ((S'\,\mathbf{sup}\,E')\,\mathbf{mod}\,M'). \end{array}$$

It should also be possible to define multiple inheritance in a similar way, with the fusion operator being used in the case where two inherited procedures act on the same state component.

7 Conclusions

We have investigated summation and product operators for predicate transformers and shown that they satisfy a rich set of algebraic laws. These are mostly distributivity laws that experience with other operators in the refinement calculus has shown to be useful. For example, in showing that summation preserves data refinement (Theorem 15), we made use of the fact that summation distributes through sequential composition. By showing that the summation operator is a special case of existing operators on predicate transformer, we were able to simplify many of the proofs for this operator. However, since the fusion and product operators do not appear to be definable in terms of existing predicate transformer operators, proofs for these operators were more involved.

Although some of the results presented here may be found in [21] and [15], we do have several new results: the fusion operator and it's relationship to the product operator, the relationship between the summation operator and existing refinement calculus constructs, and the distribution of summation through recursion. Also our approach is different: We develop product and summation operators for predicates and use properties of these operators when reasoning about predicate transformers, whereas Martin [15] uses a categorical construction that promotes products and coproducts in the category of functions to the category of relations and then to the category of predicate transformers, and Naumann [21] takes a more direct approach working within the category of predicate transformers.

We examined possible applications of these operators such as simultaneous execution, conjunction of specifications, late binding and inheritance. Simultaneous execution is important, for example, for various forms of action system parallel composition. The fusion operator provides a refinement-preserving way of composing requirements (specifications) which could also be viewed as a form of multiple inheritance. Late binding and inheritance are important features of object-oriented programming languages, and although our approach is limited in that we only model class-centered late binding, we believe it to be sufficiently powerful and find the rich set of associated laws encouraging.

Acknowledgements

The work reported here was carried out within the IRENE-project supported by the Academy of Finland. We are grateful to David Naumann for some very detailed comments. Xu Qiwen also made useful comments on an earlier draft.

A Proofs

LEMMA **3** *For relations* $P_1 : \Sigma_1 \leftrightarrow \Gamma_1,\ P_2 : \Sigma_2 \leftrightarrow \Gamma_2,$ $\qquad P_1 + P_2 \;=\; \phi_1^{-1}; P_1; \psi_1 \;\vee\; \phi_2^{-1}; P_1; \psi_2.$

Proof: Firstly, we have

$$\begin{array}{ll} & (\phi_1^{-1}; P_1; \psi_1)\ (\sigma, \gamma) \\ = & \quad \{\text{Relational composition}\} \end{array}$$

$$
\begin{aligned}
& (\exists \sigma_1 : \Sigma_1;\ \gamma_1 : \Gamma_1 \cdot \phi_1^{-1}(\sigma, \sigma_1) \wedge \psi_1(\gamma_1, \gamma) \wedge P_1(\sigma_1, \gamma_1)) \\
= & \qquad \{\text{Relational inverse}\} \\
& (\exists \sigma_1 : \Sigma_1;\ \gamma_1 : \Gamma_1 \cdot \phi_1^{-1}(\sigma, \sigma_1) \wedge \psi_1^{-1}(\gamma, \gamma_1) \wedge P_1(\sigma_1, \gamma_1)) \\
= & \qquad \{\phi_1^{-1}, \psi_1^{-1} \text{ partial functions}\} \\
& \sigma \in (\mathit{dom}\, \phi_1^{-1}) \wedge \gamma \in (\mathit{dom}\, \psi_1^{-1}) \wedge P_1(\phi_1^{-1}\sigma,\ \psi_1^{-1}\gamma).
\end{aligned}
$$

Similarly for $(\phi_1^{-1}; P_1; \psi_1)\ (\sigma, \gamma)$. Results then follows by Definition 4. □

LEMMA **4** *For predicates* $p_1 : \mathcal{P}\,\Sigma_1,\ p_2 : \mathcal{P}\,\Sigma_2$

$$
\overline{p_1} + \overline{p_2} \quad = \quad \overline{p_1 + p_2}.
$$

Proof: For $\sigma, \sigma' : \Sigma_1 + \Sigma_2$,

$$
\begin{aligned}
& (\overline{p_1} + \overline{p_2})\ (\sigma, \sigma') \\
= & \qquad \{\text{Definition 4 } (+)\} \\
& \sigma \in (\mathit{dom}\, \phi_1^{-1}) \wedge \sigma' \in (\mathit{dom}\, \phi_1^{-1}) \wedge \overline{p_1}(\phi_1^{-1}\sigma,\ \phi_1^{-1}\sigma')\ \vee \\
& \sigma \in (\mathit{dom}\, \phi_2^{-1}) \wedge \sigma' \in (\mathit{dom}\, \phi_2^{-1}) \wedge \overline{p_2}(\phi_2^{-1}\sigma,\ \phi_2^{-1}\sigma') \\
= & \qquad \{\text{Definition of } \overline{p}\} \\
& \sigma \in (\mathit{dom}\, \phi_1^{-1}) \wedge \sigma' \in (\mathit{dom}\, \phi_1^{-1}) \wedge (\phi_1^{-1}\sigma = \phi_1^{-1}\sigma') \wedge p_1(\phi_1^{-1}\sigma)\ \vee \\
& \sigma \in (\mathit{dom}\, \phi_2^{-1}) \wedge \sigma' \in (\mathit{dom}\, \phi_2^{-1}) \wedge (\phi_2^{-1}\sigma = \phi_2^{-1}\sigma') \wedge p_2(\phi_2^{-1}\sigma) \\
= & \qquad \{\phi_1, \phi_2 \text{ injective}\} \\
& \sigma \in (\mathit{dom}\, \phi_1^{-1}) \wedge \sigma' \in (\mathit{dom}\, \phi_1^{-1}) \wedge \sigma = \sigma' \wedge p_1(\phi_1^{-1}\sigma)\ \vee \\
& \sigma \in (\mathit{dom}\, \phi_2^{-1}) \wedge \sigma' \in (\mathit{dom}\, \phi_2^{-1}) \wedge \sigma = \sigma' \wedge p_2(\phi_2^{-1}\sigma) \\
= & \quad \sigma = \sigma' \wedge (\sigma \in (\mathit{dom}\, \phi_1^{-1}) \wedge p_1(\phi_1^{-1}\sigma)\ \vee\ \sigma \in (\mathit{dom}\, \phi_2^{-1}) \wedge p_2(\phi_2^{-1}\sigma)) \\
= & \qquad \{\text{Definition 1 } (+)\} \\
& \sigma = \sigma'\ \wedge\ (p_1 + p_2)\ \sigma \\
= & \quad \overline{(p_1 + p_2)}\ (\sigma, \sigma').
\end{aligned}
$$

□

THEOREM **20** *The fusion operator (Definition 8) preserves monotonicity, top homomorphism, and conjunctivity.*

Proof: Monotonicity:

$$
\begin{aligned}
(S_1 \odot S_2)\ q \quad & = \quad (\exists q_1, q_2 : \mathcal{P}\,\Gamma \mid q_1 \wedge q_2 \leq q \cdot S_1 q_1\ \wedge\ S_2 q_2) \\
& \leq \qquad \{\ q \leq q'\ \} \\
& \qquad\ (\exists q_1, q_2 : \mathcal{P}\,\Gamma \mid q_1 \wedge q_2 \leq q' \cdot S_1 q_1\ \wedge\ S_2 q_2) \\
& = \quad (S_1 \odot S_2)\ q'.
\end{aligned}
$$

Note that $S_1 \odot S_2$ is always monotonic, regardless of whether S_1 or S_2 are.

Top homomorphism: assume $S_1 \top = \top$ and $S_2 \top = \top$, then

$$
\begin{aligned}
(S_1 \odot S_2)\top \quad & = \quad (\exists q_1, q_2 : \mathcal{P}\,\Gamma \mid q_1 \wedge q_2 \leq \top \cdot S_1 q_1\ \wedge\ S_2 q_2) \\
& \geq \qquad \{\text{take } q_1 = \top,\ q_2 = \top\} \\
& \qquad\ S_1 \top\ \wedge\ S_2 \top \\
& = \quad \top.
\end{aligned}
$$

Conjunctivity:

$$
\begin{aligned}
(\forall i : I \cdot (S_1 \odot S_2)\ p_i) \quad & = \quad (\forall i : I \cdot (\exists q, q' : \mathcal{P}\,\Gamma \mid q \wedge q' \leq p_i \cdot S_1 q\ \wedge\ S_2 q')) \\
& \leq \qquad \{\text{Skolem function}\}
\end{aligned}
$$

$$
\begin{array}{ll}
 & (\exists Q, Q' : I \rightarrow \mathcal{P}\,\Gamma \cdot (\forall i : I \cdot (Q_i \wedge Q'_i \leq p_i) \;\wedge\; S_1 Q_i \;\wedge\; S_2 Q'_i)) \\
\leq & \qquad \{\text{calculus}\} \\
 & (\exists Q, Q' : I \rightarrow \mathcal{P}\,\Gamma \;\mid (\forall i : I \cdot Q_i) \wedge (\forall i : I \cdot Q'_i) \leq (\forall i : I \cdot p_i) \cdot \\
 & \qquad (\forall i : I \cdot S_1\ Q_i) \wedge (\forall i : I \cdot S_2\ Q'_i)) \\
\leq & \qquad \{S_1,S_1 \text{ meet homomorphic}\} \\
 & (\exists Q, Q' : I \rightarrow \mathcal{P}\,\Gamma \;\mid (\forall i : I \cdot Q_i) \wedge (\forall i : I \cdot Q'_i) \leq (\forall i : I \cdot p_i) \cdot \\
 & \qquad S_1(\forall i : I \cdot Q_i) \;\wedge\; S_2(\forall i : I \cdot Q'_i)) \\
\leq & \qquad \{\text{take } q = (\forall i : I \cdot Q_i),\ q' = (\forall i : I \cdot Q'_i)\} \\
 & (\exists q, q' : \mathcal{P}\,\Gamma \mid q \wedge q' \leq (\forall i : I \cdot p_i) \cdot S_1 q \;\wedge\; S_2 q') \\
= & (S_1 \odot S_2)\ (\forall i : I \cdot p_i).
\end{array}
$$

The reverse follows by monotonicity. □

THEOREM **21** *If either S_1 or S_2 is disjunctive then $S_1 \odot S_2$ is disjunctive. Similarly for continuity.*

Proof Disjunctivity:

$$
\begin{array}{ll}
 & (S_1 \odot S_2)\ (\exists i : I \cdot p_i) \\
= & (\exists q, q' : \mathcal{P}\,\Gamma \mid q \wedge q' \leq (\exists i : I \cdot p_i) \cdot S_1 q \;\wedge\; S_2 q') \\
= & \qquad \{\text{1-pt rule}\} \\
 & (\exists Q : I \rightarrow \mathcal{P}\,\Gamma;\ q, q' : \mathcal{P}\,\Gamma \mid q \wedge q' \leq (\exists i : I \cdot p_i) \;\wedge\; (\forall i : I \cdot Q_i = (q \wedge (q' \Rightarrow p_i))) \cdot S_1 q \;\wedge\; S_2 q') \\
\leq & \qquad \left\{ \begin{array}{lll} q & \leq & q \wedge q' \;\vee\; q \wedge \neg\, q' \\ & \leq & q \wedge (\exists i : I \cdot p_i) \;\vee\; Q_i \\ & \leq & (\exists i : I \cdot q \wedge p_i) \;\vee\; (\exists i : I \cdot Q_i) \\ & \leq & (\exists i : I \cdot Q_i). \end{array} \right\} \\
 & (\exists Q : I \rightarrow \mathcal{P}\,\Gamma;\ q, q' : \mathcal{P}\,\Gamma \mid (\forall i : I \cdot Q_i = (q \wedge (q' \Rightarrow p_i))) \cdot S_1(\exists i : I \cdot Q_i) \;\wedge\; S_2 q') \\
\leq & \qquad \left\{ \begin{array}{lll} Q_i \wedge q' & = & q \wedge (q' \Rightarrow p_i)) \wedge q' \\ & \leq & p_i. \end{array} \right\} \\
 & (\exists Q : I \rightarrow \mathcal{P}\,\Gamma;\ q' : \mathcal{P}\,\Gamma \mid (\forall i : I \cdot (Q_i \wedge q') \leq p_i) \cdot S_1(\exists i : I \cdot Q_i) \;\wedge\; S_2 q') \\
\leq & \qquad \{S_1 \text{ disjunctive}\} \\
 & (\exists Q : I \rightarrow \mathcal{P}\,\Gamma;\ q' : \mathcal{P}\,\Gamma \mid (\forall i : I \cdot (Q_i \wedge q') \leq p_i) \cdot (\exists i : I \cdot S_1 Q_i) \;\wedge\; S_2 q') \\
\leq & \qquad \{\text{distribution}\} \\
 & (\exists Q : I \rightarrow \mathcal{P}\,\Gamma;\ q' : \mathcal{P}\,\Gamma;\ i : I \mid (\forall i : I \cdot (Q_i \wedge q') \leq p_i) \cdot S_1 Q_i \;\wedge\; S_2 q') \\
\leq & \qquad \{\text{take } q = Q_i\} \\
 & (\exists i : I;\ q, q' : \mathcal{P}\,\Gamma \mid q \wedge q' \leq p_i \cdot S_1 q \;\wedge\; S_2 q') \\
\leq & (\exists i : I \cdot (S_1 \odot S_2)\ p_i).
\end{array}
$$

The reverse follows by monotonicity. For continuity, the proof is the same since if $(\exists i : I \cdot p_i)$ is a chain, then $(\exists i : I \cdot Q_i)$ is a chain. □

THEOREM **23** *For monotonic predicate transformer S,*

$$S \odot havoc = S.$$

Proof:

$$
\begin{array}{rcl}
(S \odot havoc)\ q & = & (\exists q_1, q_2 : \mathcal{P}\,\Gamma \mid q_1 \wedge q_2 \leq q \cdot S\ q_1 \;\wedge\; [\lambda\sigma, \gamma \cdot true]\ q_2) \\
 & = & \qquad \{\text{definition of demonic update statement}\} \\
 & & (\exists q_1, q_2 : \mathcal{P}\,\Gamma \mid q_1 \wedge q_2 \leq q \cdot S\ q_1 \;\wedge\; (\lambda\sigma \cdot q_2 = \top)) \\
 & = & \qquad \{\text{1-pt rule}\} \\
 & & (\exists q_1 : \mathcal{P}\,\Gamma \mid q_1 \leq q \cdot S\ q_1) \\
 & = & \qquad \{S \text{ monotonic}\} \\
 & & S\ q
\end{array}
$$

$\square$

THEOREM **24**

(ii) $(S_1 \wedge S_1') \odot S_2 = (S_1 \odot S_2) \wedge (S_1' \odot S_2)$ *and* $S_1 \odot (S_2 \wedge S_2') = (S_1 \odot S_2) \wedge (S_1 \odot S_2')$.

Proof: $(S_1 \wedge S_1') \odot S_2 \ \leq \ (S_1 \odot S_2) \wedge (S_1' \odot S_2)$ is trivial. We show the reverse here:

$$
\begin{array}{ll}
 & (S_1 \odot S_2) \wedge (S_1' \odot S_2)\ q \\
= & (\exists q_1, q_2 : \mathcal{P}\,\Gamma \,|\, q_1 \wedge q_2 \leq q \cdot S_1 q_1 \ \wedge \ S_2 q_2) \ \wedge \ (\exists r_1, r_2 : \mathcal{P}\,\Gamma \,|\, r_1 \wedge r_2 \leq q \cdot S_1' r_1 \ \wedge \ S_2 r_2) \\
= & (\exists q_1, q_2, r_1, r_2 : \mathcal{P}\,\Gamma \,|(q_1 \wedge q_2 \leq q) \ \wedge \ (r_1 \wedge r_2 \leq q) \cdot S_1 q_1 \ \wedge \ S_2 q_2 \ \wedge \ S_1' r_1 \ \wedge \ S_2 r_2) \\
\leq & \qquad \{\text{monotonicity of } S_1, S_1',\ \text{conjunctivity of } S_2\} \\
 & (\exists q_1, q_2, r_1, r_2 : \mathcal{P}\,\Gamma \,|(q_1 \wedge q_2 \leq q) \ \wedge \ (r_1 \wedge r_2 \leq q) \cdot S_1(q_1 \vee r_1) \ \wedge \ S_2(q_2 \wedge r_2) \ \wedge \ S_1'(q_1 \vee r_1)) \\
\leq & (\exists q_1, q_2, r_1, r_2 : \mathcal{P}\,\Gamma \,|((q_1 \vee r_1) \wedge q_2) \leq q \cdot S_1(q_1 \vee r_1) \ \wedge \ S_2(q_2 \wedge r_2) \ \wedge \ S_1'(q_1 \vee r_1)) \\
\leq & \qquad \{\text{take } q_1' = q_1 \vee r_1 \text{ and } q_2' = q_2 \wedge r_2\} \\
 & (\exists q_1', q_2' : \mathcal{P}\,\Gamma \,|\, q_1' \wedge q_2 \leq q \cdot S_1 q_1' \ \wedge \ S_2 q_2' \ \wedge \ S_1' q_1') \\
= & (S_1 \wedge S_1') \odot S_2\ q
\end{array}
$$

$\square$

THEOREM **25** *For state relations* $P, Q : \Sigma \leftrightarrow \Gamma$, *and state predicates* $p, q : \mathcal{P}\,\Sigma$,

$$\{p\};[P] \ \odot \ \{q\};[Q] \quad = \quad \{p \wedge q\};[P \wedge Q].$$

Proof: showing that $(\{p\};[P] \ \odot \ \{q\};[Q]) \ \leq \{p \wedge q\};[P \wedge Q]$ is straightforward. The reverse is more involved, and we show it here. For $r : \mathcal{P}\,\Gamma,\ \ \sigma : \Sigma$,

$$
\begin{array}{ll}
 & \{p \wedge q\};[P \wedge Q]\ r\ \sigma \\
= & \qquad \{\text{by definition}\} \\
 & p\ \sigma \wedge q\ \sigma \wedge (\forall \sigma' : \Gamma \cdot (P\ \sigma\ \sigma') \wedge (Q\ \sigma\ \sigma') \Rightarrow r\ \sigma') \\
= & p\ \sigma \wedge q\ \sigma\ \ \wedge ((\lambda \sigma' : \Gamma \cdot P\ \sigma\ \sigma')\ \wedge (\lambda \sigma' : \Gamma \cdot Q\ \sigma\ \sigma')\ \leq\ r) \\
= & \qquad \{ \quad \begin{array}{ll} & \{p\};[P]\ (\lambda \sigma' : \Gamma \cdot P\ \sigma\ \sigma')\ \sigma \\ = & p\ \sigma \wedge (\forall \sigma' : \Gamma \cdot (P\ \sigma\ \sigma') \Rightarrow (\lambda \sigma' : \Gamma \cdot P\ \sigma\ \sigma')\ \sigma') \\ = & p\ \sigma \wedge (\forall \sigma' : \Gamma \cdot (P\ \sigma\ \sigma') \Rightarrow (P\ \sigma\ \sigma')) \\ = & p\ \sigma \end{array} \quad \} \\
 & \{p\};[P]\ (\lambda \sigma' : \Gamma \cdot P\ \sigma\ \sigma')\ \sigma\ \ \wedge \\
 & \{q\};[Q]\ (\lambda \sigma' : \Gamma \cdot Q\ \sigma\ \sigma')\ \sigma\ \ \wedge \\
 & ((\lambda \sigma' : \Gamma \cdot P\ \sigma\ \sigma') \wedge (\lambda \sigma' : \Gamma \cdot Q\ \sigma\ \sigma')\ \leq\ r) \\
\Rightarrow & \qquad \{\text{take } q_1 = (\lambda \sigma' : \Gamma \cdot P\ \sigma\ \sigma'),\ q_2 = (\lambda \sigma' : \Gamma \cdot Q\ \sigma\ \sigma')\ \} \\
 & (\exists q_1, q_1 : \mathcal{P}\,\Gamma \cdot (q_1 \wedge q_2\ \leq\ r)\ \wedge\ \{p\};[P]\ q_1\ \sigma\ \wedge\ \{q\};[Q]\ q_2\ \sigma) \\
= & (\ \{p\};[P]\ \odot\ \{q\};[Q]\ )\ r\ \sigma.
\end{array}
$$

$\square$

THEOREM **30** *The product operator preserves monotonicity and top and bottom homomorphism, as well as conjunctivity and disjunctivity.*

Proof: of preservation of disjunctivity [21]: preservation of distribution over the empty join follows from preservation of bottom homomorphism. Let $(\exists i : I \cdot q_i)$ be a nonempty join. If $q_i = \perp$ for each $i \in I$. Then

$$
\begin{array}{rcl}
(S_1 \otimes S_2)\ (\exists i : I \cdot q_i) & = & (S_1 \otimes S_2)\ \perp \\
 & = & (\exists i : I \cdot (S_1 \otimes S_2)\ \ \perp) \\
 & = & (\exists i : I \cdot (S_1 \otimes S_2)\ q_i).
\end{array}
$$

Otherwise we may assume, without loss of generality, that I is nonempty and $q_i \neq \perp$ for each $i \in I$. Then

$$
\begin{array}{ll}
 & (\exists i : I \cdot (S_1 \otimes S_2)\ q_i) \\
= & (\exists i : I;\ r_1 : \mathcal{P}\,\Gamma_1;\ r_2 : \mathcal{P}\,\Gamma_2 \mid r_1 \times r_2 \leq q_i \cdot S_1 r_1\ \wedge\ S_2 r_2) \\
\geq & (\exists i : I;\ r_1 : \mathcal{P}\,\Gamma_1;\ r_2 : \mathcal{P}\,\Gamma_2 \mid (r_1 \times r_2 \leq q_i)\ \wedge\ r_1 \neq \perp\ \wedge r_2 \neq \perp \cdot S_1 r_1\ \wedge\ S_2 r_2) \\
= & \qquad \{\text{see Lemma 8 below}\} \\
 & (\exists i : I;\ r_1 : \mathcal{P}\,\Gamma_1;\ r_2 : \mathcal{P}\,\Gamma_2;\ \sigma_1 : \Gamma_1;\ \sigma_2 : \Gamma_2 \mid \\
 & \qquad (r_1 \times r_2 \leq q_i)\ \wedge\ r_1 \neq \perp\ \wedge r_2 \neq \perp\ \wedge r_1\sigma_1\ \wedge\ r_2\sigma_2\ \cdot \\
 & \qquad S_1(\lambda\sigma_1' : \Gamma_1 \cdot \sigma_1' = \sigma_1)\ \wedge\ S_2(\lambda\sigma_2' : \Gamma_2 \cdot \sigma_2' = \sigma_2)) \\
= & \qquad \{\text{see ** below, } q_i \neq \perp\ \} \\
 & (\exists i : I;\ \sigma_1 : \Gamma_1;\ \sigma_2 : \Gamma_2 \mid q_i(\sigma_1, \sigma_2)\ \cdot \\
 & \qquad S_1(\lambda\sigma_1' : \Gamma_1 \cdot \sigma_1' = \sigma_1)\ \wedge\ S_2(\lambda\sigma_2' : \Gamma_2 \cdot \sigma_2' = \sigma_2)) \\
= & (\exists \sigma_1 : \Gamma_1;\ \sigma_2 : \Gamma_2 \mid (\exists i : I \cdot q_i)(\sigma_1, \sigma_2)\ \cdot \\
 & \qquad S_1(\lambda\sigma_1' : \Gamma_1 \cdot \sigma_1' = \sigma_1)\ \wedge\ S_2(\lambda\sigma_2' : \Gamma_2 \cdot \sigma_2' = \sigma_2)) \\
= & \qquad \{\text{see ** below, } (\exists i : I \cdot q_i) \neq \perp\ \} \\
 & (\exists r_1 : \mathcal{P}\,\Gamma_1;\ r_2 : \mathcal{P}\,\Gamma_2;\ \sigma_1 : \Gamma_1;\ \sigma_2 : \Gamma_2 \mid \\
 & \qquad (r_1 \times r_2 \leq (\exists i : I \cdot q_i))\ \wedge\ r_1 \neq \perp\ \wedge r_2 \neq \perp\ \wedge r_1\sigma_1\ \wedge\ r_2\sigma_2\ \cdot \\
 & \qquad S_1(\lambda\sigma_1' : \Gamma_1 \cdot \sigma_1' = \sigma_1)\ \wedge\ S_2(\lambda\sigma_2' : \Gamma_2 \cdot \sigma_2' = \sigma_2)) \\
= & \qquad \{\text{see Lemma 8 below}\} \\
 & (\exists q_1 : \mathcal{P}\,\Gamma_1;\ q_2 : \mathcal{P}\,\Gamma_2 \mid (r_1 \times r_2 \leq (\exists i : I \cdot q_i)) \cdot S_1 r_1\ \wedge\ S_2 r_2) \\
= & (S_1 \otimes S_2)\ (\exists i : I \cdot q_i).
\end{array}
$$

** These steps are possible because, for $q \neq \perp$,

$$q(\sigma_1, \sigma_2) \quad = \quad (\exists r_1, r_2 \cdot (r_1 \times r_2 \leq q)\ \wedge\ r_1 \neq \perp\ \wedge r_2 \neq \perp\ \wedge r_1\sigma_1\ \wedge\ r_2\sigma_2).$$

These steps would not be possible if the same strategy was attempted on the fusion operator. □

Lemma 8 *If $S : \Gamma \longmapsto \Sigma$ is disjunctive, then for $q : \mathcal{P}\,\Gamma$, where $q \neq \perp$,*

$$S\ q \quad = \quad (\exists \sigma : \Gamma \mid q\ \sigma \cdot S\ (\lambda\sigma' : \Gamma \cdot \sigma' = \sigma)).$$

Proof:

$$
\begin{array}{lll}
(\exists \sigma : \Gamma \mid q\ \sigma \cdot S\ (\lambda\sigma' : \Gamma \cdot \sigma' = \sigma)) & = & \qquad \{\text{disjunctivity}\} \\
 & & S\ (\exists \sigma : \Gamma \mid q\ \sigma \cdot (\lambda\sigma' : \Gamma \cdot \sigma' = \sigma)) \\
 & = & \qquad \{\text{1-pt rule}\} \\
 & & S\ q.
\end{array}
$$

□

References

[1] R.J.R. Back. *Correctness Preserving Program Refinements: Proof Theory and Applications.* Tract 131, Mathematisch Centrum, Amsterdam, 1980.

[2] R.J.R. Back and R. Kurki-Suonio. Decentralisation of process nets with centralised control. In *2nd ACM SIGACT-SIGOPS Symp. on Principles of Distributed Computing*, pages 131–142, 1983.

[3] R.J.R. Back and K. Sere. Superposition refinement of parallel algorithms. In K.A. Parker and G.A. Rose, editors, *FORTE'91*. North–Holland, 1992.

[4] R.J.R. Back and J. von Wright. Refinement concepts formalised in higher order logic. *Formal Aspects of Computing*, 5:247–272, 1990.

[5] R.J.R. Back and J. von Wright. *Refinement Calculus.* Book in preparation, Åbo Akademi, 1994.

[6] M.J. Butler. *A CSP Approach To Action Systems.* D.Phil. Thesis, Programming Research Group, Oxford University, 1992.

[7] M.J. Butler. *Refinement and Decomposition of Value-Passing Action Systems.* In E. Best, editor, *CONCUR'93*, volume LNCS 715. Springer–Verlag, 1993.

[8] E.W. Dijkstra. *A Discipline of Programming.* Prentice-Hall, 1976.

[9] P.H.B. Gardiner and C.C. Morgan. Data refinement of predicate transformers. *Theoretical Computer Science*, 87:143–162, 1991.

[10] J. Gutknecht. *Object-Oriented Programming with Oberon.* Lecture Notes from Eastern Finland Universities International Summer School on Novel Computing, Lapeenranta University of Technology, Finland, 1994.

[11] J. He, C.A.R. Hoare, and J.W. Sanders. Data refinement refined. In *European Symposium on Programming*, volume LNCS 213. Springer–Verlag, 1986.

[12] P. Hitchcock and D. Park. Induction rules and termination proofs. In *IRIA Conference on Automata Languages and Programming Theory*, France, 1972.

[13] C.A.R. Hoare. *Communicating Sequential Processes.* Prentice–Hall, 1985.

[14] C.B. Jones. *Systematic Software Development using VDM – Second Edition.* Prentice–Hall, 1990.

[15] C.E. Martin. *Preordered Categories and Predicate Transformers.* D.Phil. Thesis, Programming Research Group, Oxford University, 1991.

[16] C.C. Morgan. *Programming from Specifications.* Prentice–Hall, 1990.

[17] C.C. Morgan. The cuppest capjunctive capping, and Galois. In A.W. Roscoe, editor, *A Classical Mind: Essays in Honour of C.A.R. Hoare.* Prentice–Hall, 1994.

[18] C.C. Morgan, K.A. Robinson, and P.H.B. Gardiner. *On the Refinement Calculus.* Technical Monograph PRG–70, Programming Research Group, Oxford University, October 1988.

[19] J.M. Morris. A theoretical basis for stepwise refinement and the programming calculus. *Sci. Comp. Prog.*, 9(3):298–306, 1987.

[20] H.P. Mossenboeck. *Object-Oriented Programming in Oberon-2.* Springer-Verlag, 1994.

[21] D.A. Naumann. *Two-Categories and Program Structure: Data Types, Refinement Calculi, and Predicate Transformers.* Ph.D. Thesis, University of Texas at Austin, 1992.

[22] D.A. Naumann. *On the Essence of Oberon.* Southwestern University, Georgetown, Texas, 1993.

[23] G. Nelson. A generalization of Dijkstra's calculus. *ACM Trans. Prog. Lang. Sys.*, 11(4):517–561, 1989.

[24] J.M. Spivey. *The Z Notation - A Reference Manual.* Prentice–Hall, 1989.

[25] B.M. Utting and K. Robinson. Modular reasoning in an object-oriented refinement calculus. In *Mathematics of Program Construction, 1992*, volume LNCS 669. Springer–Verlag, 1993.

[26] J. von Wright. The lattice of data refinement. *Acta Informatica*, 31(2):105–135, 1994.

[27] N. Ward. Adding specification constructs to the refinement calculus. In *FME'93*, volume LNCS 670. Springer–Verlag, 1993.