16

# A Generalization of Dijkstra's Calculus

by Greg Nelson

April 2, 1987


digital

**Systems Research Center**
130 Lytton Avenue
Palo Alto, California 94301

# Systems Research Center

DEC's business and technology objectives require a strong research program. The Systems Research Center and two other corporate research laboratories are committed to filling that need.

SRC opened its doors in 1984. We are still making plans and building foundations for our long-term mission, which is to design, build, and use new digital systems five to ten years before they become commonplace. We aim to advance both the state of knowledge and the state of the art.

SRC will create and use real systems in order to investigate their properties. Interesting systems are too complex to be evaluated purely in the abstract. Our strategy is to build prototypes, use them as daily tools, and feed the experience back into the design of better tools and the development of more relevant theories. Most of the major advances in information systems have come through this strategy, including time-sharing, the ArpaNet, and distributed personal computing.

During the next several years SRC will explore high-performance personal computing, distributed computing, communications, databases, programming environments, system-building tools, design automation, specification technology, and tightly coupled multiprocessors.

SRC will also do work of a more formal and mathematical flavor; some of us will be constructing theories, developing algorithms, and proving theorems as well as designing systems and writing programs. Some of our work will be in established fields of theoretical computer science, such as the analysis of algorithms, computational geometry, and logics of programming. We also expect to explore new ground motivated by problems that arise in our systems research.

DEC is committed to open research. The Company values the improved understanding that comes with widespread exposure and testing of new ideas within the research community. SRC will therefore freely report results in conferences, in professional journals, and in our research report series. We will seek users for our prototype systems among those with whom we have common research interests, and we will encourage collaboration with university researchers.

Robert W. Taylor, Director

# Corrections to SRC–16

*Greg Nelson*
DEC *Systems Research Center*
*130 Lytton Avenue*
*Palo Alto, CA 94301*

*June 5, 1987*

Edsger Dijkstra has pointed out two errors in "A generalization of Dijkstra's calculus" (SRC–16). In the proof on page 42 that $A\,;B$ is continuous in $B$, I assumed without justification that $\mathrm{wp}(A, ?)$ was $\vee$-continuous (blush). And in the proof on page 44 that $[\![x \mid A]\!]$ is continuous in $A$, I distributed $\forall$ over $\vee$ (blush, blush).

In fact these operators are not continuous. However, they are monotonic, and the main results of SRC–16 on recursion and tail recursion can be proved using monotonicity only, as this note will show. Familiarity with SRC–16 will be assumed.

First the counterexamples showing that composition and projection are not continuous. Let $A$ and $B_i$ (for $i \geq 0$) be $n$-commands defined by

$$A \equiv \text{set } n \text{ to any natural number}$$
$$B_i \equiv n \leq i \;\rightarrow\; n := 0 \boxtimes \mathit{Loop}$$

Then it is easy to check that

$$A\,;B_i \;\equiv\; n := 0 \;\Box\; \mathit{Loop} \quad \text{for all } i$$
$$(\sqcup i :: B_i) \;\equiv\; n := 0$$

and therefore that

$$
\begin{aligned}
& A\,;(\sqcup i :: B_i) \\
\equiv\;& A\,;n := 0 \\
\equiv\;& n := 0 \\
\not\equiv\;& \mathit{Loop} \;\Box\; n := 0 \\
\equiv\;& (\sqcup i :: \mathit{Loop} \;\Box\; n := 0) \\
\equiv\;& (\sqcup i :: A\,;B_i)
\end{aligned}
$$

and similarly that

$$
\begin{aligned}
& [\![n \mid (\sqcup i :: Bi)]\!] \\
\equiv\;& [\![n \mid n := 0]\!]
\end{aligned}
$$

$$\begin{array}{ll} \equiv & \mathit{Skip} \\ \not\equiv & \mathit{Skip} \;\square\; \mathit{Loop} \\ \equiv & (\sqcup i :: \mathit{Skip} \;\square\; \mathit{Loop}) \\ \equiv & (\sqcup i :: [\![ n \mid Bi ]\!]) \end{array}$$

Thus $[\![ n \mid B ]\!]$ is not continuous in $B$, and neither is $A\,;B$ if $A$ is unboundedly non-deterministic.

Here is the corrected theorem on continuity and monotonicity:

**Corrected Theorem 7.** *The fundamental operators and projection enjoy the following continuity and monotonicity properties:*

| | |
|---|---|
| $P \to A$ | continuous in $A$ |
| $A \;\square\; B$ | continuous in $A$ and $B$ |
| $A \boxtimes B$ | continuous in $A$ and $B$ |
| $A\,;B$ | continuous in $A$; monotonic in $B$ |
| $A\,;B$ | continuous in $B$ if $\mathrm{wp}(A, ?)$ is $\vee$-continuous |
| $[\![ x \mid A ]\!]$ | monotonic in $A$ |

The proofs in SRC–16 establish all the continuity claims. The two monotonicity claims follow from the following lemma.

**Lemma 4.** Let $f$ be a command transformer (that is, a map from commands to commands) and $h$ and $hl$ predicate transformers such that for any command $A$ and predicate $R$:

$$\mathrm{w(l)p}(f(A),\ R) \equiv h(l)(\mathrm{w(l)p}(A, R))\,.$$

Then $f$ is $\sqsubseteq$-monotonic.

*Proof.* Note first that $h$ and $hl$ must be conjunctive, since

$$\begin{array}{ll} & h(l)(?) \\ \equiv & h(l)(\mathrm{w(l)p}(\mathit{Skip},\ ?)) \\ \equiv & \mathrm{w(l)p}(f(\mathit{Skip}),\ ?) \end{array}$$

and $f(\mathit{Skip})$ being by assumption a command, its predicate transformers are conjunctive.

Now observe that

$$\begin{array}{ll} & A \sqsubseteq B \\ \equiv & (\forall R : \mathrm{w(l)p}(A, R) \Rightarrow\!(\Leftarrow)\ \mathrm{w(l)p}(B, R)) \end{array}$$

$\Rightarrow$ {$h$ and $hl$ are monotonic}

$(\forall R : h(l)(\mathrm{w(l)p}(A, R)) \Rightarrow(\Leftarrow) h(l)(\mathrm{w(l)p}(B, R)))$

$\equiv (\forall R : \mathrm{w(l)p}(f(A), R) \Rightarrow(\Leftarrow) \mathrm{w(l)p}(f(B), R))$

$\equiv f(A) \sqsubseteq f(B)$

Thus $f$ is $\sqsubseteq$-monotonic. ∎

The proofs of the monotonicity claims of the corrected theorem follow from Lemma 4 and the semantic equations

$$\mathrm{w(l)p}(A\,;B,\ R) \equiv \mathrm{w(l)p}(A, \mathrm{w(l)p}(B, R))$$
$$\mathrm{w(l)p}([\![x \mid A]\!], R) \equiv (\forall x : \mathrm{w(l)p}(A, R))$$

Lemma 4 with $\mathrm{w(l)p}(A, ?)$ for $h(l)$ establishes that $A\,;B$ is monotonic in $B$, and with $(\forall x : ?)$ for $h(l)$ establishes that $[\![x \mid A]\!]$ is monotonic in $A$. ∎

Since the operators on commands are only monotonic, not continuous, the appeal in the proof of Theorem 8 to the Limit Theorem must be replaced by an appeal to the Generalized Limit Theorem of Hitchcock and Park.

The generalized theorem asserts that the least fixpoint of a monotonic function $f$ is equal to $f^\alpha$, for some ordinal $\alpha$, where $f^\alpha$ is defined by the inductive rule

$$f^\alpha = (\sqcup \beta : \beta < \alpha : f(f^\beta))\,.$$

See "The generalized limit theorem" (CGN46) for more details. This change is reflected in an obvious way in the statement of Theorem 8.

Finally, these changes to the Limit Theorem and to Theorem 8 require a change to the proof of Theorem 9 on tail recursion. Starting in the middle of page 47, the proof should read as follows:

To show that $\mathrm{wp}(X_0, R)$ is the strongest fixpoint, observe that since $f$ is monotonic, Theorem 8 implies that $X_0 = f^\alpha$ for some ordinal $\alpha$. Let $P$ be any fixpoint of $g(?, R)$; we prove by induction that $\mathrm{wp}(f^\alpha, R) \Rightarrow P$ for all $\alpha$. Therefore $\mathrm{wp}(X_0, R) \Rightarrow P$, in other words, $\mathrm{wp}(X_0, R)$ is the strongest fixpoint. The induction is easy:

$\mathrm{wp}(f^\alpha, R) \Rightarrow P$

$\equiv \mathrm{wp}((\sqcup \beta : \beta < \alpha : f(f^\beta)), R) \Rightarrow P$

$\equiv (\vee\, \beta : \beta < \alpha : \mathrm{wp}(f(f^\beta), R)) \Rightarrow P$

$\equiv (\forall\, \beta : \beta < \alpha : \mathrm{wp}(f(f^\beta), R) \Rightarrow P)$

$\equiv$ {$P$ is a fixpoint of $g(?, R)$}

$$
\begin{array}{ll}
 & (\forall\, \beta : \beta < \alpha : \mathrm{wp}(f(f^{\beta}),\, R) \Rightarrow g(P,R)) \\
\equiv & \{\text{Tail Recursion identity}\} \\
 & (\forall\, \beta : \beta < \alpha : g(\mathrm{wp}(f^{\beta},\, R),\, R) \Rightarrow g(P,R)) \\
\Leftarrow & \{g \text{ is monotonic}\} \\
 & (\forall\, \beta : \beta < \alpha : \mathrm{wp}(f^{\beta}, R) \Rightarrow P) \\
\equiv & \{\text{induction}\} \\
 & \mathbf{TRUE}
\end{array}
$$

The proof that $\mathrm{wlp}(X_0, R)$ is the weakest fixpoint is so similar that I won't write it out.

# The generalized limit theorem

*Greg Nelson*
DEC *Systems Research Center*
*130 Lytton Avenue*
*Palo Alto, CA 94301*

*June 5, 1987*

The following theorem is well known:

**Limit Theorem.** *If $f$ is a continuous function on a partially-ordered set in which every chain has a join, then $f$ has a least fixpoint given by*

$$(\sqcup n : n \geq 0 : f^n(\min))\,,$$

*where* min *denotes the minimum element—which must exist, since it is the join of the empty chain.*

Hitchcock and Park [1] have improved this theorem, essentially by weakening the constraint on $f$ from "continuous" to "monotonic" and extending the range of $n$ from "the integers" to "the ordinals". Since I find that I need their generalized limit theorem in my treatment of recursion in Dijkstra's calculus, I am recording in this note my version of their proof.

The reader is assumed to be familiar with the following facts about ordinals:

1. An ordinal is a total well-founded order.
2. For ordinals $\alpha$ and $\beta$, the ordinal $\alpha + \beta$ is $\alpha$ followed by $\beta$.
3. For ordinals $\alpha$ and $\beta$, $\alpha \leq \beta$ means $\alpha$ is isomorphic to a prefix of $\beta$.
4. The relation $\leq$ on ordinals is itself a total well-founded order.
5. There exist ordinals of arbitrarily large cardinality.

As a consequence of 4, induction is valid over the ordinals. (Induction on ordinals is traditionally called "transfinite induction"—a phrase that makes me think of vampires.) Point 5 is a consequence of (and is equivalent to) the Axiom of Choice.

Let $f$ be a monotonic function on a partially ordered set in which every chain has a join. For each ordinal $\alpha$, we define an element $f^\alpha$ of the partially ordered set by the rule

$$f^\alpha = (\sqcup \beta : \beta < \alpha : f(f^\beta))\,. \tag{1}$$

Note that $f^\alpha$ is defined in terms of $f^\beta$ for $\beta < \alpha$, so the definition is not circular. In particular,

$$\begin{aligned} f^0 &= \text{empty join} = \min \\ f^1 &= f(f^0) \\ f^2 &= f(f^0) \sqcup f(f^1) = f(f^1) \\ &\vdots \\ f^\omega &= f(f^0) \sqcup f(f^1) \sqcup \ldots \\ &= f^0 \sqcup f^1 \sqcup f^2 \sqcup \ldots \end{aligned}$$

(The ordinal $\omega$ is the first infinite ordinal, namely the order type of the natural numbers.)

Note that we have not attempted to define $f^\alpha(x)$ for all $\alpha$ and $x$, since this will not be possible in general. (In particular, if $x$ and $f(x)$ are incomparable.)

I have modified Hitchcock and Park's theorem slightly: they define

$$\begin{aligned} &f^0 = \min \\ &f^\alpha = f(\sqcup \beta : \beta < \alpha : f^\beta) \ \text{ for } \alpha \neq 0 \end{aligned}$$

Their definition gives the wrong value (intuitively speaking) for limit ordinals, and introduces an unnecessary case analysis.

Of course we have not yet shown that the join in (1) exists. This will be one of a series of little results that lead to the generalized limit theorem. The series is essentially the same as in Hitchcock and Park's proof.

**Lemma 1.** If $f^\alpha$ and $f^\beta$ are defined and $\alpha < \beta$, then $f^\alpha \sqsubseteq f^\beta$.

*Proof.*

$$\begin{aligned} & f^\alpha \\ = \ & (\sqcup \gamma : \gamma < \alpha : f(f^\gamma)) \\ \sqsubseteq \ & \{\text{enlarging the range increases the join}\} \\ & (\sqcup \gamma : \gamma < \beta : f(f^\gamma)) \\ = \ & f^\beta \quad \blacksquare \end{aligned}$$

**Lemma 2.** $f^\alpha$ is defined for all $\alpha$.

*Proof.* By induction on $\alpha$:

$$\begin{aligned} & f^\beta \text{ is defined for } \beta < \alpha \\ \Rightarrow \ & \{\text{Lemma 1, totality of} \leq \text{on ordinals}\} \end{aligned}$$

$$
\begin{array}{ll}
 & f^\beta \text{ and } f^{\beta'} \text{ defined and comparable for } \beta, \beta' < \alpha \\
\Rightarrow & \{f \text{ monotonic}\} \\
 & f(f^\beta) \text{ and } f(f^{\beta'}) \text{ defined and comparable for } \beta, \beta' < \alpha \\
\Rightarrow & \{(1), \text{joins of chains exist}\} \\
 & f^\alpha \text{ is defined} \quad \blacksquare
\end{array}
$$

**Lemma 3.** $f^{\alpha+1} = f(f^\alpha)$ for all $\alpha$.

*Proof.*

$$
\begin{array}{ll}
 & f^{\alpha+1} \\
= & (\sqcup \beta : \beta < \alpha + 1 : f(f^\beta)) \\
= & f(f^\alpha) \sqcup (\sqcup \beta : \beta < \alpha : f(f^\beta)) \\
= & \{\sqcup \text{ distributes over } \sqcup\} \\
 & (\sqcup \beta : \beta < \alpha : f(f^\alpha) \sqcup f(f^\beta)) \\
= & \{\text{Lemma 1, monotonicity of } f\} \\
 & (\sqcup \beta : \beta < \alpha : f(f^\alpha)) \\
= & f(f^\alpha) \quad \blacksquare
\end{array}
$$

**Lemma 4.** For any fixpoint $x$ of $f$ and any ordinal $\alpha$, we have $f^\alpha \sqsubseteq x$.

*Proof.* By induction on $\alpha$.

$$
\begin{array}{ll}
 & (\forall \beta : \beta < \alpha : f^\beta \sqsubseteq x) \\
\Rightarrow & \{\text{monotonicity of } f\} \\
 & (\forall \beta : \beta < \alpha : f(f^\beta) \sqsubseteq f(x)) \\
= & \{x \text{ is a fixpoint of } f\} \\
 & (\forall \beta : \beta < \alpha : f(f^\beta) \sqsubseteq x) \\
\Rightarrow & \{\text{join is the least upper bound}\} \\
 & (\sqcup \beta : \beta < \alpha : f(f^\beta)) \sqsubseteq x \\
= & \{(1)\} \\
 & f^\alpha \sqsubseteq x \quad \blacksquare
\end{array}
$$

**Lemma 5.** If $\alpha < \beta$ and $f^\alpha = f^\beta$, then the common value of $f^\alpha$ and $f^\beta$ is the least fixpoint of $f$.

*Proof.* Suppose $f^\alpha = f^\beta$. Then

$$
\begin{array}{ll}
& \alpha < \beta \\
\Rightarrow & \alpha \leq \alpha + 1 \leq \beta \\
\Rightarrow & \{\text{Lemma 1}\} \\
& f^\alpha \sqsubseteq f^{\alpha+1} \sqsubseteq f^\beta \\
\Rightarrow & \{f^\alpha = f^\beta\} \\
& f^\alpha = f^{\alpha+1} \\
\Rightarrow & \{\text{Lemma 3}\} \\
& f^\alpha \text{ is a fixpoint of } f
\end{array}
$$

By Lemma 4, $f^\alpha \sqsubseteq x$ for any other fixpoint $x$, so $f^\alpha$ is the least fixpoint. ∎

**Generalized Limit Theorem.** *If $f$ is a monotonic function on a partially ordered set in which every chain has a join, then $f$ has a least fixpoint given by $f^\alpha$, for some ordinal $\alpha$.*

*Proof.* Let $\gamma$ be an ordinal whose cardinality exceeds the cardinality of the partially ordered set. Then we must have $f^\alpha = f^\beta$ for some $\alpha < \beta < \gamma$. By Lemma 5, $f^\alpha$ is the least fixpoint of $f$. ∎

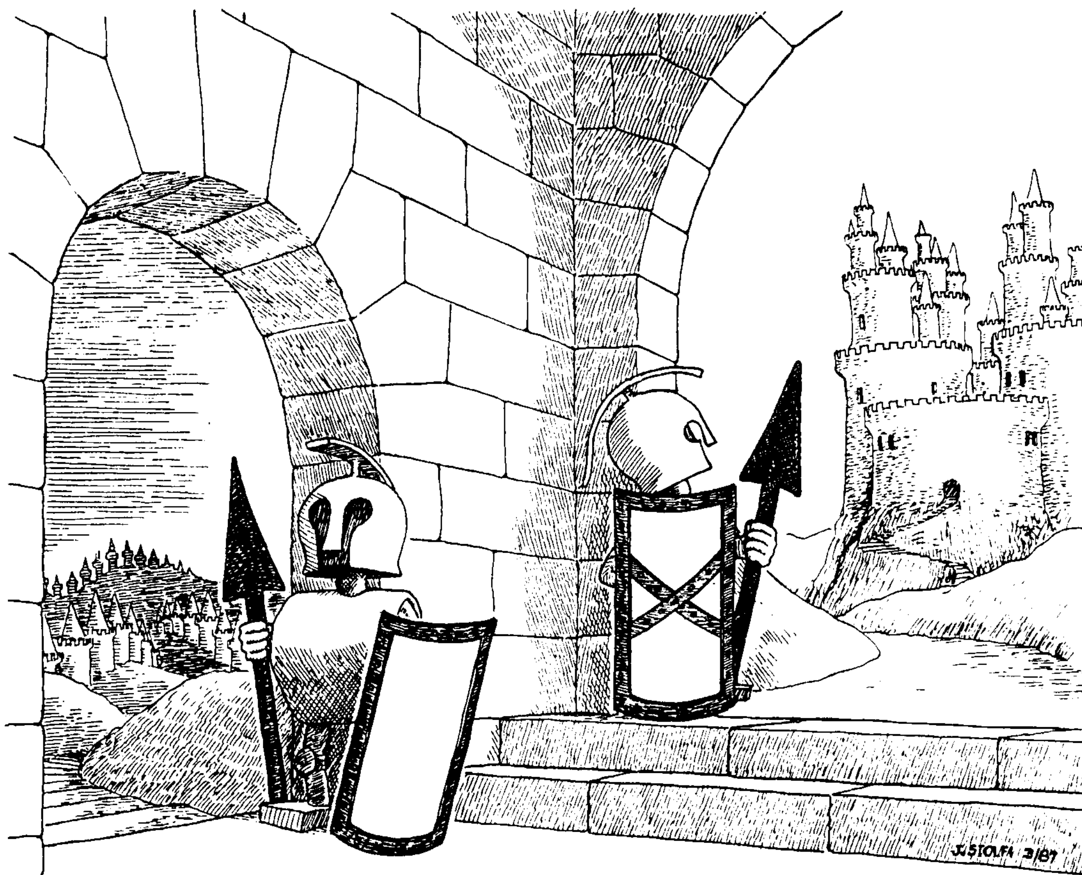
## Reference

[1] Peter Hitchcock and David Park. Induction Rules and Termination Proofs. *IRIA Conf. Autom. Lang. & Program. Theory.* France, 1972.

# A Generalization of Dijkstra's Calculus

Greg Nelson

April 2, 1987



©Digital Equipment Corporation 1986

This work may not be copied or reproduced in whole or in part for any commercial purpose. Permission to copy in whole or in part without payment of fee is granted for nonprofit educational and research purposes provided that all such whole or partial copies include the following: a notice that such copying is by permission of the Systems Research Center of Digital Equipment Corporation in Palo Alto, California; an acknowledgment of the authors and individual contributors to the work; and all applicable portions of the copyright notice. Copying, reproducing, or republishing for any other purpose shall require a license with payment of fee to the Systems Research Center. All rights reserved.

**Author's abstract**

This paper gives a self-contained account of a general calculus of program semantics, from first principles through the semantics of recursion. The calculus is like Dijkstra's guarded commands, but without the Law of the Excluded Miracle; like extended dynamic logic, but with a different approximation relation; like a relational calculus studied by deBakker, but with partial relations as well as total relations; like predicative programming, but with a more standard notion of total correctness. The treatment of recursion uses the fixpoint method from denotational semantics.

Greg Nelson

**Capsule review**

This report describes several technical advances in axiomatic semantics. Sections one through seven present a gentle introduction to an extended version of Dijkstra's wp-calculus. The primary technical innovations are the introduction of partial commands, and the complete characterization of partial commands as predicates, following and extending Hehner's work. The final three chapters begin the synthesis of axiomatic and denotational semantics by presenting the first coherent treatment of general recursion for an axiomatic semantics. As an example of the power of this characterization, the semantics of tail-recursion (and thus the primitive control structures of Dijkstra's calculus) are derived.

Although partial commands are not executable, they are an important part of this work. They simplify many of the equations, and reduce the number of primitive operations. The principal drawback to adopting partial commands is that Dijkstra's Law of the Excluded Miracle must be abandoned, thereby eliminating one of the better theorem names from computer science.

Mark Manasse

## Contents


| | | |
|---|---|---|
| | Introduction | 1 |
| 1 | A brief survey of semantic theory | 4 |
| 2 | Commands as relations | 8 |
| 3 | Commands as operations | 11 |
| 4 | Preliminaries, mostly about predicates | 14 |
| 5 | Commands as predicate transformers | 17 |
| 6 | Variables | 22 |
| 7 | Commands as predicates | 27 |
| 8 | Recursion | 33 |
| 9 | Iteration and tail-recursion | 45 |
| 10 | Concluding remarks | 49 |
| | Acknowledgements | 50 |
| | References | 51 |
| | Index | 53 |


## ERRATA

**Errors on pages 42 and 44 are corrected in the accompanying notes.**

## Introduction

There is a remarkable omission from the guarded command language that Edsger Dijkstra introduced in his classic *Discipline of Programming*: it doesn't have procedures. One of the reasons that he left them out was to avoid getting tangled in the complexities of recursion, as we can infer from this remark in his preface: "the semantics of a repetitive construct can be defined in terms of a recurrence relation between *predicates*, whereas the semantic definition of a general recursion requires a recurrence relation between *predicate transformers*. This shows quite clearly why I regard general recursion as an order of magnitude more complicated than just repetition" [2].

Dijkstra's remark went against the spirit of the time, which was all for treating iteration as a special case of recursion. A sample of the reaction was E. C. R. Hehner's paper "**do** considered **od**" [10], which criticized Dijkstra's repetitive control structure and argued that recursion should be used instead. But, although the paper doesn't mention it, Hehner's treatment of recursion works only for tail recursions, not for general recursions. The technical difficulty mentioned in Dijkstra's remark has remained an obstacle to the treatment of general recursion.

Recent developments suggest a way to get around the difficulty: Hehner and C. A. R. Hoare have approached programs as predicates instead of predicate transformers, and Dijkstra and others have derived the correspondence between the two approaches. I call this the *cograph correspondence*. This paper grew out of my effort to handle general recursion in Dijkstra's calculus by using the cograph correspondence to reduce predicate transformer recurrences to predicate recurrences. The effort produced the solution described in sections 7 and 8.

The solution uses the method of fixpoint semantics: define an approximation relation on guarded commands and prove that the operators of the language distribute over joins of chains in this relation. This is the method that Dana Scott used to construct models of the lambda-calculus, laying the foundations of domain theory and denotational semantics.

The heart of the solution is the definition of an appropriate approximation relation. The definition vindicates two distinctive features of guarded commands: nondeterminism and total correctness. At first these features seem problematical, but in the end they force a definition that is unexpectedly simple and symmetric.

The definition is in section 8; here is a preview for readers who are familiar with Dijkstra's calculus. For commands $A$ and $B$, let $A \sqsubseteq B$ (read *$A$ approximates $B$*) mean that for any $R$,

$$\mathrm{wp}(A,R) \Rightarrow \mathrm{wp}(B,R) \quad \text{and} \quad \mathrm{wlp}(B,R) \Rightarrow \mathrm{wlp}(A,R)\,.$$

Under this relation, the set of guarded commands forms a *domain* in the sense of denotational semantics; that is, a continuous partial order.

As I worked out the details of this solution, I became more and more dissatisfied with Dijkstra's well-known Law of the Excluded Miracle. The Law states that any command $A$ satisfies $\mathrm{wp}(A, \text{FALSE}) = \text{FALSE}$; in other words, no command can establish the postcondition FALSE. This seems hard to argue with, but I found that the Law was never useful, and it complicated the cograph correspondence. As we shall see, the technical effect of the Law is to exclude *partial* commands; that is, commands that correspond to partial relations between states and outcomes. It is reasonable to prohibit partial commands from practical general-purpose programming languages, because in general they require backtracking to implement. Also, they are tricky to think about, because their semantics hinge on subtle aspects of nondeterminism. But these are no reasons to prohibit them from the semantic calculus, any more than we should prohibit fractions on the grounds that whole numbers are easier to compute with and understand. In fact, partial commands provide a useful framework when nondeterminism arises in practice, as in the parsing of context-free languages.

I concluded that the Law of the Excluded Miracle is like bubble sort: it is unfortunate that it has a catchy name, because the world would be better off to forget it. I have dropped the Law and admitted partial commands, obtaining a general version of Dijkstra's calculus that is in many ways simpler than his classical version, just as the rational numbers have simpler algebraic properties than the integers.

The general calculus includes all the control structure operators of the classical version. But to avoid violating the Law of the Excluded Miracle, the classical syntax restricts the use of these operators; for example,

$$\textbf{if } P \to A \;\square\; Q \to B \textbf{ fi}$$

is allowed, but

$$(A \;\square\; B)\,;(P \to C)$$

is not. The general calculus allows the operators to be combined freely. This is a step forward, since, as a rule, operators are good and syntax is bad. For example, the syntactic freedom simplifies proofs by structural induction. A less fundamental difference is that the general calculus includes a new operator $\boxtimes$, a non-commutative version of $\square$, which is needed in the tail recursions that define **if** $A$ **fi** and **do** $A$ **od**.

The plan for this paper is to give a self-contained account of the general calculus, from first principles through the semantics of recursion. The

first half of the paper focuses on the nature of partial commands; the last half on recursion. Technicalities are postponed to the second half whenever possible. The only prerequisite for reading the paper is the ability to follow manipulations involving boolean operators and quantifiers.

Section 1 is an essay that introduces the various technical approaches and semantic models that arise in axiomatic semantics. In sections 2 through 5, we begin the study of the general calculus by considering the semantics of the fundamental operations for sequencing and testing. To avoid the confusion that surrounds partial commands, we will approach the general calculus via several of the techniques explained in section 1: as a calculus of relations, as a calculus of operations, and as a calculus of predicate transformers, examining the role of partial commands each time. In section 6 we consider the semantics of assignment and local variables, which leads to the cograph correspondence in section 7. In section 8 we consider general recursion, and in section 9 the special case of tail recursion and iteration.

The first half of the paper seeks to develop intuition by approaching the calculus in several ways, but in the latter half of the paper the predicate transformer approach is taken as basic. People seem to grasp the relational approach more readily, but serious work goes faster with predicate transformers. In practice, the ingredients of program verification are programs and predicates; it is therefore desirable to have syntactic rules for manipulating them directly, without translating in and out of the language of relations. In fact we will take the most extreme form of the predicate transformer approach, the axiomatic approach. This means that we will rely only on the algebraic properties of predicates and not on the definition of a predicate as a boolean-valued function on a state space. As a consequence, all of the paper's results hold in abstract algebras of predicates, such as those introduced by Halmos and by Tarski and his colleagues.

This is not sterile abstraction: the axiomatic approach is simpler than approaches that are cluttered with unnecessary assumptions. Its only disadvantage is that its definitions often seem to be pulled out of a hat—the definition of the approximation relation is a good example. I will try to counteract this by motivating each axiomatic definition in the framework of the more concrete relational approach. This lengthens the paper, but the emphasis seems well-placed, since finding the definitions is the hard part: once they are right, the theorems prove themselves.

In addition to Dijkstra, Hehner, Hoare, and Scott, I would like to mention the intellectual debts that this work owes to J. W. deBakker, H. Egli, David Harel, and Vaughan Pratt. Egli [6] and deBakker [1] studied an approximation relation that is essentially similar to $\sqsubseteq$. However, their systems

did not include partial commands. Also, they did not discover the symmetrical definition of $\sqsubseteq$ in terms of wp and wlp, probably because they approached semantics via relations instead of via predicate transformers, and the definition of $\sqsubseteq$ in terms of relations obscures the underlying symmetry. Harel and Pratt [8, 9] defined a version of Dynamic Logic that contains the same collection of commands as my generalization of the wp-calculus. But they did not discover the approximation relation at all; their treatment of recursion uses the subset relation instead. As we shall see, this unnecessarily limits the recursions that can be handled.

## 1 A brief survey of semantic theory

This section presents a brief taxonomy of programming logics, initially approaching them via relations.

The relational approach is connected to the ordinary operational approach by the following rule: Command $A$ relates state $s$ to outcome $o$ if $o$ is a possible outcome of activating $A$ from initial state $s$. Two subtle choices are left open in this apparently simple rule. First, the set of outcomes may be the same as the set of states, or it may contain an additional element to represent the outcome of infinite looping. Second, the relations may be restricted to total relations (those in which each state is related to at least one outcome), or partial relations may be admitted. If both the looping outcome and partial relations are excluded, then there is no way to represent a looping computation, so this model is uninteresting. The other three models have all been thoroughly explored.

The earliest model to be explored included partial relations but excluded the looping outcome. A command that is certain to loop forever from some input state is modelled by relating that state to no outcome. A problem with this approach is that it cannot model a command that might loop or halt from the same input state. The practical consequence is that the model is useless for proving termination, so we will call it the *partial correctness model.* For example, Dynamic Logic (or DL) used this model.

A natural way to avoid the problem is to use what we will call the *total correctness model*, which includes the looping outcome and excludes partial relations. This allows an input state to be related only to the looping outcome, only to proper (that is, non-looping) outcomes, or to both the looping outcome and to some proper outcomes. Note that partial relations are excluded from the total correctness model: in this model each input state is related to at least one outcome, on the grounds that *something* must happen when the command is activated in the input state. (For simplicity, we will lump run-time errors together with infinite loops.) Dijkstra presented

his calculus of guarded commands without an explicit relational model, but (as we shall see) the relational model consistent with his calculus is the total correctness model. David Parnas's limited domain relations are a technical variation on the total correctness model [18].

David Harel and Vaughan Pratt were the first to realize the utility of the model that includes both partial relations and the looping outcome, which we will call the *general model* [8, 9]. They showed that within DL, the way to reach the general model from the partial correctness model is to add another elementary predicate transformer $[\alpha]^+$ to the previous $[\alpha]$. In this paper, we follow the lead of Harel and Pratt by accepting the general model. We will see that the way to reach the general model from Dijkstra's total correctness model (which already has two elementary predicate transformers, wp and wlp) is to drop the Law of the Excluded Miracle.

The general model raises the question of the operational interpretation of partial commands. That is, if a command $A$ relates an input state $s$ to no outcome at all, what does it mean to activate $A$ in input state $s$? This will be explained in section 3.

In our taxonomy, we have so far approached the different semantic models via relations. But each of them can also be approached via predicate transformers or Hoare logic. Thus there are two dimensions of variation among semantic systems: semantic model and technical approach. Let us explore the other technical approaches to the partial correctness model.

Hoare logic is based on primitive assertions of the form $\{P\}A\{Q\}$, where $P$ and $Q$ are predicates and $A$ is a command, which means operationally that if $A$ is activated in a state where $P$ is true, then $Q$ is true of any state in which $A$ might halt. The connection to the relational approach is obvious: the assertion means that for all states $s$ and $s'$, if $A$ relates $s$ to $s'$, then $P(s)$ implies $Q(s')$. Thus $A$ as a relation between predicates is determined by $A$ as a relation between states, and, as is easily shown, the converse holds as well.

The approach to the partial correctness model via predicate transformers goes as follows. For command $A$ and predicate $Q$, define $\mathrm{wlp}(A, Q)$ to be the weakest $P$ such that $\{P\}A\{Q\}$; that is, the weakest precondition sufficient to ensure the postcondition $Q$. Thus the predicate transformer $\mathrm{wlp}(A, ?)$ is determined by $A$ as a relation on predicates. (We will write $\mathrm{wlp}(A, ?)$ to denote the map $P \mapsto \mathrm{wlp}(A, P)$.) The converse holds as well, since $\{P\}A\{Q\}$ is equivalent to $P \Rightarrow \mathrm{wlp}(A, Q)$.

Therefore, in developing the partial correctness model, commands can be considered either as relations on states, as relations on predicates, or as predicate transformers, the choice of approach being one of technique. The

same choices are available for the total correctness model and the general model, though the looping outcome complicates things somewhat. For example, if predicate transformers are used, the effect of the looping outcome is that a command is determined by two predicate transformers (wp and wlp) instead of one. The general model requires no new apparatus—the same two predicate transformers suffice. It simply differs from the total correctness model by having more commands, namely those commands that violate the Law of the Excluded Miracle.

Recently E. C. R. Hehner [11] and C. A. R. Hoare [13] have explored a system called *predicative programming*, which involves both a new approach and a new model. We will make use of their approach, but not their model. In this approach, a command is considered as a predicate on a state space that includes variables representing both the initial and final states of the computation. For example, the command $x := x + 1$ is identified with the predicate $\acute{x} = \grave{x} + 1$, where $\grave{x}$ and $\acute{x}$ are the initial and final value of $x$, respectively. This "commands-as-predicates" approach is still another technical variation on the other approaches, since there is a natural correspondence between predicates and conjunctive predicate transformers (which are the only kind of predicate transformers that matter in this context). A pretty proof of the correspondence was presented by the Eindhoven Tuesday Afternoon Club [4]. We will call this the *cograph correspondence*, and make use of it to reduce predicate transformer recurrences to predicate recurrences.

In addition to the "commands as predicates" approach, predicative programming involves a new model, which we will call the HH model, after Hehner and Hoare. The HH model is based on the principle that if you don't care what the outcome of a computation is, then you don't care whether it ever terminates. Counter-examples may come to mind (for example, flipping a coin), but the principle still suggests a model with at least technical interest. Let us approach the model via relations, as we did with the other models. The HH model discards the looping outcome; replacing it by the convention that if a command relates an initial state to *every* final state, it is considered that infinite looping is also a possible outcome; if even a single final state is excluded, then infinite looping is forbidden. The HH model has aroused some controversy, a sample of which may be found in the recent exchange between Parnas and Hehner in the CACM [19].

Our taxonomy is summarized in Table 1. The plan for this paper is to drop the Law of the Excluded Miracle from Dijkstra's calculus, and thereby arrive at the same place that Harel and Pratt arrived at by adding $[\alpha]^+$ to Dynamic Logic. In the resulting system, we will solve recurrence relations by using an approximation relation that generalizes the one introduced by

deBakker and Egli to partial commands. As a technical device, we will use the cograph correspondence to pass back and forth between predicate transformers and predicates.

| Technical Approach | Semantic Model | | | |
|---|---|---|---|---|
| | Partial | Total | General | HH |
| Relations on Predicates | Original Hoare Logic | | | |
| Relations on States | Early Relational Models | Parnas' LDR[1]; Egli/Debakker | DL$^+$ | |
| Predicate Transformers | DL | wp-calculus with LEM[2] | DL$^+$; wp-calculus w/o LEM | |
| Predicates | | | Cographs | Predicative Programming |

[1] LDR ≡ Limited Domain Relations
[2] LEM ≡ Law of the Excluded Miracle

**Table 1.** Various calculi of programming semantics, classified in rows by technical approach and in columns by the nature of their semantic model.

The area we have just surveyed is called "axiomatic semantics". Since a neighboring research area is called "denotational semantics", you might think that axiomatic semantics is not denotational. Not so: all the approaches and all the models are "denotational" in the sense that the meaning of a command is a mathematical object and the meaning of an expression is a function of the meanings of its operands. A better characterization of the difference between the two areas is that historically, axiomatic semantics has addressed the problem of change of state, while denotational semantics has addressed the problem of higher order functions. Axiomatic semantics has made the assignment statement mathematically respectable; denotational semantics has made the lambda-calculus mathematically respectable. (The problem of the assignment statement has also been attacked under the banner of denotational semantics, but the results are rather operational, and separable

from the underlying domain theory. At least, so it appears to me.) It is unfortunate that the names of these areas represent their relationship so poorly, since the resulting confusion delays their synthesis, which would be a promising foundation for programming theory. Following Hehner and deBakker, and more recently Dijkstra, this work is aimed towards such a synthesis.

## 2 Commands as relations

We will begin by approaching the general calculus by way of relations. The connection to the operational approach is given by the rule

Command $A$ relates state $s$ to outcome $o$
$\equiv$ $o$ is a possible outcome of activating $A$ in initial state $s$

An outcome is either a state, or the "looping outcome" $\perp$, which represents infinite looping and run-time errors. Partial relations are allowed.

In order to give examples of this connection it will be helpful to have a few conventional operational definitions. The *alternative construct*

$$\textbf{if } P_1 \rightarrow A_1 \ \square \ \ldots \ \square \ P_n \rightarrow A_n \textbf{ fi}$$

means: select some $P_i$ that is true and execute $A_i$. If all the $P$'s are false, the outcome is defined to be $\perp$. The *repetitive construct*

$$\textbf{do } P_1 \rightarrow A_1 \ \square \ \ldots \ \square \ P_n \rightarrow A_n \textbf{ od}$$

means: repeatedly select some $P_i$ that is true and execute $A_i$, until all the $P$'s are false. If the repetition never halts, the outcome is defined to be $\perp$.

For example, in the lower right of Figure 1 is an illustration of the relation associated with the command

$$\textbf{if } x = 0 \rightarrow \mathit{Skip} \ \square \ y = 0 \rightarrow y := 1 \textbf{ fi},$$

in which $x$ and $y$ are boolean variables. (The command *Skip* is a no-op.) To explain the remainder of the figure, we need some more relational definitions.

We will write $\mathcal{G}(A)$ to denote the *guard* of $A$, that is, the domain of the relation $A$ considered as a predicate:

$$\mathcal{G}(A)s \equiv (\exists o : Aso)\,.$$

If $\mathcal{G}(A) \equiv \text{TRUE}$, we say that $A$ is *total*. Otherwise $A$ is *partial*.

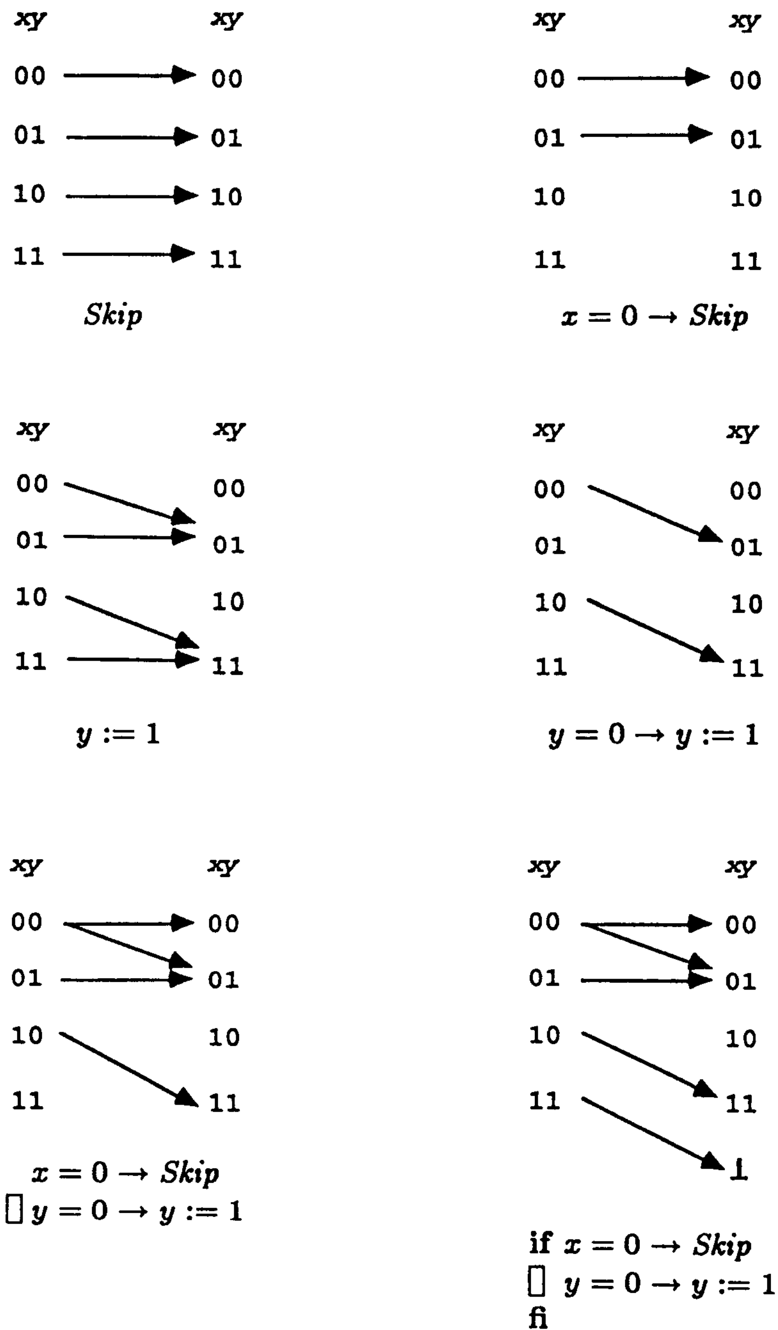



**Figure 1.** The anatomy of a guarded command. The command in the lower right is composed of the subcommands shown in the rest of the figure.

Let $A$ and $B$ denote commands, $P$ a predicate on states, $s$ a state, and $o$ an outcome. Then the relational definitions of the operators appearing in Figure 1 are

$$Skip\, so \equiv s = o$$

$$(P \rightarrow A)so \equiv Ps \wedge Aso$$

$$(A \,\square\, B)so \equiv Aso \vee Bso$$

$$(\textbf{if } A \textbf{ fi})so \equiv Aso \vee (\neg\, \mathcal{G}(A)s \wedge o = \bot)$$

That is, the no-op *Skip* relates each state to itself. The outcomes of $P \rightarrow A$ are the outcomes of $A$ from initial states satisfying $P$. The outcomes of $A \,\square\, B$ are the outcomes of $A$ and of $B$. The command **if** $A$ **fi** is a total command formed from $A$ by adding a looping outcome from all states that have no other outcome. The illustration in Figure 1 of the anatomy of an alternative construct should now be clear.

In the classical calculus, which admits only total commands, **if**–**fi** is a $2n$-ary operator, and $\rightarrow$ and $\square$ do not have the status of operators at all. Partial commands are technically attractive, since they allow $\rightarrow$ and $\square$ to be binary operators, and **if**–**fi** to be a unary operator.

Here are the relational definitions of some other important operators:

$$Loop\, so \equiv o = \bot$$

$$Fail\, so \equiv \text{FALSE}$$

$$(A \boxtimes B)so \equiv Aso \vee (Bso \wedge \neg\, \mathcal{G}(A)s)$$

$$(A\, ;B)so \equiv (\exists s' : Ass' \wedge Bs'o) \vee (Aso \wedge o = \bot)$$

That is, *Loop* is the command that relates each state to the looping outcome; *Fail* is the command that relates no state to any outcome; $A \boxtimes B$ is similar to $A \,\square\, B$, but the outcomes of $B$ are included only from those initial states from which $A$ has no outcome; and $A\, ; B$ is the composition of $A$ and $B$ together with all looping outcomes of $A$. Note that *Fail* is an identity for the operators $\square$ and $\boxtimes$, and *Skip* is an identity for the operator ;. The importance of the unconventional *Fail* and $\boxtimes$ will become clear later.

An apology for unconventional nomenclature: There are some circumstances in which it is desirable to extend the space of outcomes still further, by distinguishing between the looping outcome and other outcomes that represent runtime errors (for example, in the compiler verification by Manasse and Nelson [15]). To avoid confusion in those circumstances, we will write *Loop* instead of the conventional *Abort.*

We will call $\rightarrow$, $\Box$, $\boxtimes$, and ; the *fundamental operators* because all the common control structures can be defined in terms of them by using recursion. For example, we shall see that **do** $A$ **od** is the $\sqsubseteq$-least solution of

$$X : X \equiv (A ; X) \boxtimes Skip .$$

We won't consider the relational definition of **do** $A$ **od**, since it is awkward and, in view of the definitions to be presented in subsequent sections, unnecessary.

In order of decreasing binding power, the operators are $; \quad \rightarrow \quad \Box \quad \boxtimes$. For example,

$$A \Box P \rightarrow B ; C \boxtimes D \quad \text{means} \quad (A \Box (P \rightarrow (B ; C))) \boxtimes D .$$

At first it seems that the operator $\boxtimes$ is mildly unattractive because it is asymmetric, but a little reflection brings the realization that it is extremely unattractive because it is not monotonic. That is, adding outcomes to $A$ can remove outcomes from $A \boxtimes B$ by masking outcomes of $B$, so the operator is not monotonic with respect to the subset relation. Since monotonicity is essential to the treatment of recursion, this is far worse than asymmetry. My first reaction was to try to do without $\boxtimes$. But this is hopeless, since, as we shall see, **do–od** isn't $\subseteq$-monotonic either! We can't do without **do**, so there is no alternative but to abandon the subset relation and find a new relation with respect to which all the operators are monotonic. As we shall see, the approximation relation $\sqsubseteq$ does the trick.

## 3 Commands as operations

Partial commands are technically attractive, but their operational interpretation is somewhat subtle. What would happen if the command $x = 0 \rightarrow Skip$ were executed in a state where $x \neq 0$? If nothing happens, what is the difference between the given command and *Skip*? If an error happens, what is the difference between the given command and **if** $x = 0 \rightarrow Skip$ **fi**? Something else must happen, and the only possibility is to backtrack, as this section explains in detail.

The operational approach is based on the following scenario: given a digital machine at rest in some initial state, the activation of a command produces a sequence of state transitions, the outcome of which is to halt in some state or to loop forever. There are two important points to discuss: First, the outcome need not be functionally determined by the initial state—that is, the command may be nondeterministic. Second, associated with each command is a predicate called its guard, and a command can only be activated in states where its guard is true.

"But", you ask, "what would happen if a command somehow got activated in a state where its guard is false?". This is not a particularly useful question. Imagine commanding a typist to press the space bar, then release it, then release it again. In assigning a meaning to this command, it is no help to ask, "but what would happen if the typist somehow did manage to release the space bar a second time?".

But people will ask the question anyway, since the anthropomorphic overtones of operational thinking lead us to imagine the machine "trying" to activate a command whose guard is false. So as not to be without an answer, let us say that such an attempt *fails*, whereas if the guard is true then activation *succeeds*. Note that if a command cannot be activated, the attempt to do so will fail *without changing the state of the computation.* If any computation is involved in the attempt to activate, it must have no side effects.

"But", you ask, "what happens if a true guard leads to a computation that has side effects and then runs into a false guard?". The answer is that the implementation must not let this happen. The explanation of this somewhat unsatisfying answer brings us to the other important point, nondeterminism.

To avoid confusion, it is usually wise to qualify the word "nondeterminism" with one of the adjectives "blind" or "clairvoyant". An implementation of blind nondeterminism is allowed to choose blindly between the alternative execution paths. For example, if a programming language reference manual declares that arguments to procedures may be evaluated in any order, it is warning the programmer that the implementation reserves the right to be blindly nondeterministic. In contrast, an implementation of clairvoyant nondeterminism is required to choose an execution that "succeeds", for some notion of success. For example, in automata theory, a successful computation is one that accepts its input, and an implementation of a nondeterministic automaton is required to find an accepting computation if one exists.

Applying these general remarks to the case at hand, we require that the implementation of our commands be clairvoyant enough to find a computation that succeeds, but allow it to choose blindly between all successful computations. That is, the implementation must make its nondeterministic choices in such a way that no command is ever activated in a state where its guard is false. For example, imagine a typist commanded to press either the space bar or the shift key and then release the space bar, in an initial state with all keys up. The typist must be clairvoyant enough to press the space bar rather than the shift key, since the space bar can only be released when it is down.

So much for the general framework. Here are the operational definitions

of the fundamental operators, together with **do–od** and **if–fi**:

| | |
|---|---|
| $A ; B$ | activate $A$, then activate $B$ |
| $A \,\square\, B$ | activate either $A$ or $B$ |
| $A \boxtimes B$ | activate $A$ if possible, else activate $B$ |
| $P \rightarrow A$ | activate $A$ from a state where $P$ is true |
| **do** $A$ **od** | activate $A$ until it fails |
| **if** $A$ **fi** | activate $A$ until it succeeds |

There are several consequences of these definitions worth noting.

First, note that FALSE $\rightarrow A$ is independent of $A$; it is the unique command whose guard is false, which cannot be activated in any state. This command was introduced earlier as *Fail.*

The command **do** *Fail* **od**, which activates *Fail* until it fails, is therefore a no-op. This command was introduced earlier as *Skip.*

The command **if** *Fail* **fi**, which activates *Fail* until it succeeds, evidently activates *Fail* forever, and therefore the outcome of this command is to loop forever. This command was introduced earlier as *Loop.* An equally valid operational definition of *Loop* is **do** *Skip* **od**.

It is now clear that **do–od** is not $\subseteq$-monotonic, since *Fail* $\subseteq$ *Skip*, but **do** *Fail* **od** $\not\subseteq$ **do** *Skip* **od**, since *Skip* $\not\subseteq$ *Loop.*

Since a failed attempt to activate a command has no effect on the state, the definition "Activate $A$ until it succeeds" for **if** $A$ **fi** is a little odd. It would have been more natural to write "Activate $A$ unless it fails, in which case loop". But if there is more than one process executing concurrently, these two definitions are not equivalent. Although we will not consider concurrency in this paper, the definition of **if-fi** is phrased to accommodate it. In a concurrent program, **if** $P \rightarrow A$ **fi** has the semantics of a "blocking if" that waits for $P$ to be true and then executes $A$. In a sequential program, the wait will last forever. This interpretation was proposed by Dijkstra in his account of the Owicki-Gries theory [3].

If $B$ is partial, then the execution of $A ; B$ will in general require nontrivial clairvoyance, since a wrong choice made while executing $A$ may lead to an outcome of $A$ that doesn't satisfy the guard of $B$. In fact, it is not difficult to see that a partial command as the second argument to semicolon is essentially the only construction that requires clairvoyance on the part of the implementation. Furthermore, **do** $A$ **od** and **if** $A$ **fi** are always total, and $A ; B$ is total if $A$ and $B$ are, and similar rules hold for the other operators. Thus the need for clairvoyance can be removed by simple syntactic restrictions. The restrictions of the classical calculus are sufficient, but stricter than necessary.

As an example of the power of clairvoyant nondeterminism, we will define a command $E$ that parses simple arithmetic expressions, assuming we are given a procedure $Id$ that parses identifiers and procedures $Op_+$ and $Op_\times$ that parse the tokens for the operators $+$ and $\times$. By *parsing* a syntactic category we mean reading from the input the longest legal instance of the category, or failing if no prefix of the input belongs to the category. $E$ is defined recursively:

$$E \;\equiv\; E\,;Op_+\,;E \;\boxtimes\; E\,;Op_\times\,;E \;\boxtimes\; Id\,.$$

Note that the use of $\boxtimes$ instead of $\Box$ gives $\times$ more binding power than $+$, while leaving the associativity of the operators unspecified. This example reminds us that at least one tool for translating nondeterministic programs into efficient deterministic ones has been an engineering success: the parser generator. (It also suggests a simple way of looking at semantic attachments to grammars, in which inherited and synthesized attributes become in and out procedure parameters, respectively.)

## 4 Preliminaries, mostly about predicates

This section presents a few definitions that we will need before going into our third approach to the system, via predicate transformers.

*Functional definitions.* A *predicate* is a total boolean-valued function defined on some Cartesian product space, the elements of which will be called *states*. The symbol $\equiv$ denotes equality between predicates: $P \equiv Q$ means $P$ and $Q$ are equal as functions. The symbol $\Rightarrow$ denotes the strength order between predicates: $P \Rightarrow Q$, read "$P$ is as strong as $Q$" or "$Q$ is as weak as $P$" or "$P$ implies $Q$", means that no state is mapped to TRUE by $P$ and to FALSE by $Q$.

*Boolean properties.* It follows that $\Rightarrow$ is a reflexive partial order with a maximum element TRUE and a minimum element FALSE in which any two predicates $P$ and $Q$ have a least upper bound $P \vee Q$ and a greatest lower bound $P \wedge Q$, where $\wedge$ and $\vee$ distribute over one another, and in which every predicate $P$ has a complement $\neg P$ satisfying $P \vee \neg P \equiv$ TRUE and $P \wedge \neg P \equiv$ FALSE. These properties can be summarized by saying that the predicates form a *boolean algebra.*

In order of decreasing binding power, the operators are $\neg \quad \wedge \quad \vee$, followed by the relations $\Rightarrow$ and $\equiv$.

The existence of TRUE and FALSE depend on the fact that the state space, being a Cartesian product, is not empty (since the empty Cartesian product is the singleton containing the empty tuple). This is the only connection

between the product structure of the state space and the boolean properties of the predicates. The product structure is more important to the cylindrical properties, which will be considered next.

*Cylindrical properties.* The coordinate functions that project the state space onto its Cartesian factors are called *variables.* A predicate $P$ is *independent* of a variable $v$ (and is called a *v-cylinder*) if it has the same value at any two states that differ in the $v$ coordinate only. For any predicate $P$ there exists a predicate $(\exists v : P)$ which is the strongest $v$-cylinder that is as weak as $P$, and there exists a predicate $(\forall v : P)$ which is the weakest $v$-cylinder that is as strong as $P$.

Two variables have the same *type* if the corresponding Cartesian factors are the same set. If $u$ and $v$ are variables of the same type and $P$ is a predicate, we define $P(u : v)$ to be the predicate that "says about $v$ whatever $P$ says about $u$"; that is, $P(u : v)$ holds of state $s$ if $P$ holds of $s'$, where $s'$ is the state which coincides with $s$ in all coordinates except $u$, where it has the value $v(s)$. Another common notation for $P(u : v)$ is $P^u_v$. For example, $(u < v)(u : w) \equiv (w < v)$. A pair of the form $(u : v)$ is called a *substitution.* Substitutions are boolean algebra homomorphisms. The simple rules for composing substitutions and distributing them over quantifiers will not be listed explicitly.

The cylindrical properties of predicates can be summarized by saying that the predicates form a *polyadic algebra* in the sense of Halmos [7], a *cylindric algebra* in the sense of Henkin, Monk, and Tarski [12], or a *predicate algebra*, according to the axiomatization that I like to use [17].

*Completeness property.* Every set of predicates has a greatest lower bound under the $\Rightarrow$ order. This is easy to see: the greatest lower bound maps a state to TRUE if and only if every predicate in the set maps it to TRUE. Similarly, every set of predicates has a least upper bound.

*The axiomatic program.* In the axiomatic approach to the semantics of guarded commands, the boolean properties of predicates are used for treating the semantics of the fundamental operators, the cylindrical properties are used for variables and assignment, and completeness is used for iteration and recursion. The axiomatic view is that the definition of a predicate as a function on a state space should play no formal role. That is, a calculus of guarded commands should be definable in any complete predicate algebra, regardless of whether it has been constructed from a state space.

*Miscellaneous notations.* Except for the parenthesis convention, the following notations are due to Dijkstra.

We will use square brackets to denote the following drastic map on

predicates: $[\text{TRUE}] = \text{TRUE}$, and $[P] = \text{FALSE}$ for all other $P$.

We will write

(operation dummies : range : term)

to denote the combination via the given operation of the values assumed by the given term as the dummies vary over the given range. The operation must be commutative, associative, and (if the range is empty) possess an identity. If the range is obvious from the context, it will be omitted. For example, the greatest lower bound of the set $S$ of predicates is denoted by $(\wedge\ P : P \in S : P)$, or by $(\wedge\ P :: P)$ if $S$ is obvious from the context.

A proof of $P \equiv Q$ will often be written in the form

$$
\begin{array}{ll}
 & P \\
\equiv & \{\text{hint why } P \equiv R\} \\
 & R \\
\equiv & \{\text{hint why } R \equiv Q\} \\
 & Q
\end{array}
$$

A proof of $P \Rightarrow Q$ will be written similarly, except that some of the steps may use $\Rightarrow$ instead of $\equiv$. In both cases, unnecessary hints will be omitted.

In order to express formulas involving the two operators wp and wlp compactly, the *parenthesis convention* will be used: a formula containing parenthesized expressions represents two formulas, in one of which the parenthesized expressions are ignored, in the other of which each parenthesized expression is either inserted, or substituted for the item to its left, whichever is suggested by the context. For example, instead of the two formulas

$$\mathrm{wlp}((\sqcup\ A :: A),\ R) \equiv (\wedge\ A :: \mathrm{wlp}(A, R))$$
$$\mathrm{wp}((\sqcup\ A :: A),\ R) \equiv (\vee\ A :: \mathrm{wp}(A, R))$$

we will write the single formula

$$\mathrm{w(l)p}((\sqcup\ A :: A),\ R) \equiv (\vee(\wedge)\ A :: \mathrm{w(l)p}(A, R))\,.$$

*Fixpoint theorems.* Next we will consider two classical theorems concerning complete partial orders that play a central role in the fixpoint method.

Let $\sqsubseteq$ be a reflexive partial order on some set, and let $\sqcap$ denote meet (i.e., greatest lower bound) and $\sqcup$ denote join (i.e., least upper bound) with respect to $\sqsubseteq$. A *chain* is a subset that is totally ordered by $\sqsubseteq$. A function $f$ on the set is *monotonic* if $x \sqsubseteq y$ implies $f(x) \sqsubseteq f(y)$, and *chain-continuous*

if $f((\sqcup x :: x)) \equiv (\sqcup x :: f(x))$ whenever $x$ ranges over a non-empty chain and both sides are defined. Note that if $f$ is chain-continuous, it is monotonic:

$$
\begin{array}{ll}
 & x \sqsubseteq y \\
\equiv & \{\text{Connection between } \sqsubseteq \text{ and } \sqcup\} \\
 & (x \sqcup y) = y \\
\Rightarrow & \{\text{Equal arguments map to equal results}\} \\
 & f(x \sqcup y) = f(y) \\
\equiv & \{f \text{ is chain-continuous and } \{x,y\} \text{ is a non-empty chain}\} \\
 & (f(x) \sqcup f(y)) = f(y) \\
\equiv & \{\text{Connection between } \sqcup \text{ and } \sqsubseteq\} \\
 & f(x) \sqsubseteq f(y)
\end{array}
$$

Here are the two classical fixpoint theorems:

**Limit Theorem.** *If* $(\sqcup x :: x)$ *exists when* $x$ *ranges over any chain, and* $f$ *is chain-continuous, then* $f$ *has a least fixpoint given by*

$$(\sqcup i : i \geq 0 : f^i(\mathrm{min}))\,,$$

*where the superscript indicates iterated composition and* min *denotes the minimum element—which must exist, since it is the join of the empty chain.*

**Knaster-Tarski Theorem.** *If* $f$ *is monotonic and* $q = (\sqcap x : f(x) \sqsubseteq x : x)$ *exists, then* $q$ *is the least fixpoint of* $f$*, as well as the least solution of the equation* $x : f(x) \sqsubseteq x$.

The Limit Theorem is proved in Manna's text [16]; it is originally due to Kleene [14]. For proofs of the Knaster-Tarski Theorem, see the original reference [20] or Dijkstra's notes [5]. The standard version of the Knaster-Tarski Theorem requires that every set of elements has a meet, but we will have a use for our stronger formulation, which requires only that a particular set of elements has a meet. The proof of the standard version also works for our strengthened version. The Limit Theorem could be strengthened similarly, but we won't bother to do so since the conventional formulation is all we will need.

## 5 Commands as predicate transformers

In this section we interpret the general calculus by means of predicate transformers, which are simply maps from predicates to predicates. We consider only the semantics of the fundamental operators, and therefore rely only on the boolean properties of the predicates.

In this approach a command $A$ is defined to be a pair of predicate transformers, written $\text{wp}(A,?)$ and $\text{wlp}(A,?)$, satisfying the ***pairing condition***, which is that for any predicate $R$,

$$\text{wp}(A,R) \equiv \text{wp}(A,\text{TRUE}) \wedge \text{wlp}(A,R)\,,$$

and the ***conjunctivity condition***, which is that $\text{wlp}(A,?)$ distributes over any conjunction, and $\text{wp}(A,?)$ distributes over any non-empty conjunction.

We will motivate this definition by connecting it to the relational approach. The basic connection is

the predicate $\text{wp}(A,P)$ holds of state $s$
$\equiv$ every outcome of $A$ from $s$ is proper and satisfies $P$

the predicate $\text{wlp}(A,P)$ holds of state $s$
$\equiv$ every proper outcome of $A$ from $s$ satisfies $P$.

All predicates in this paper are predicates over states, therefore no predicate (not even TRUE) can be said to hold of $\perp$. Also, the "$s$" in the two equivalences above can be omitted, since preconditions are understood to be predicates on states. These two observations allow us to write succinctly

$\text{w(l)p}(A,R) \equiv$ every (proper) outcome of $A$ satisfies $R$.

Here are two noteworthy consequences of the basic connection:

$\text{wp}(A,\text{TRUE})(s) \equiv$ every outcome of $A$ from $s$ is proper.

$\text{wp}(A,\text{FALSE})(s)$
$\equiv$ every outcome of $A$ from $s$ satisfies FALSE
$\equiv$ there is no outcome of $A$ from $s$.

Thus $\text{wp}(A,\text{TRUE})$ characterizes those states from which error-free termination is guaranteed, while $\text{wp}(A,\text{FALSE})$ characterizes those states from which activation is impossible. Hence

$$\mathcal{G}(A) \equiv \neg\,\text{wp}(A,\text{FALSE})\,.$$

From this equation we see that Dijkstra's Law of the Excluded Miracle, $\text{wp}(A,\text{FALSE}) \equiv \text{FALSE}$, can be written $\mathcal{G}(A) \equiv \text{TRUE}$. That is, the effect of the Law is to exclude partial commands.

Here is the motivation for the pairing condition:

$\text{wp}(A,P)$
$\equiv$ every outcome of $A$ is proper and satisfies $P$

$\equiv$ every outcome of $A$ is proper and every proper outcome of $A$ satisfies $P$

$\equiv$ $\mathrm{wp}(A, \text{TRUE}) \wedge \mathrm{wlp}(A, P)$

And for the conjunctivity condition:

$\mathrm{wlp}(A, (\wedge\, i :: P_i))$

$\equiv$ every proper outcome of $A$ satisfies every $P_i$

$\equiv$ for every $i$, every proper outcome of $A$ satisfies $P_i$

$\equiv$ $(\wedge\, i :: \mathrm{wlp}(A, P_i))$

$\mathrm{wp}(A, (\wedge\, i :: P_i))$

$\equiv$ every outcome of $A$ is proper and satisfies every $P_i$

$\equiv$ {the range of $i$ is not empty}

for every $i$, every outcome of $A$ is proper and satisfies $P_i$

$\equiv$ $(\wedge\, i :: \mathrm{wp}(A, P_i))$

Thus for any relation between states and outcomes, the corresponding pair of predicate transformers satisfies the pairing and conjunctivity conditions. Conversely, given any such pair $\mathrm{wp}(A, ?)$ and $\mathrm{wlp}(A, ?)$, the relation $A$ connected with them is uniquely determined. To show this, we introduce the notation $\bar{s}$ to denote the *co-atomic* predicate that is holds of all states except the state $s$. Then

$A$ relates $s$ to the proper outcome $t$

$\equiv$ $\neg$(every proper outcome of $A$ from $s$ differs from $t$)

$\equiv$ $\neg\, \mathrm{wlp}(A, \bar{t})(s)$

$A$ relates $s$ to $\perp$

$\equiv$ $\neg$(every outcome of $A$ from $s$ is proper)

$\equiv$ $\neg\, \mathrm{wp}(A, \text{TRUE})(s)$

Thus the "proper part" of $A$ as a relation is determined by $\mathrm{wlp}(A, ?)$, and the "improper part" by $\mathrm{wp}(A, \text{TRUE})$.

So much for the motivation of the definition. Here are the definitions of the commands *Fail*, *Skip*, and *Loop*, and of the fundamental operators:

$\mathrm{w(l)p}(\mathit{Fail}, R) \equiv \text{TRUE}$

$\mathrm{w(l)p}(\mathit{Skip}, R) \equiv R$

$\mathrm{w(l)p}(\mathit{Loop}, R) \equiv \text{FALSE}\ (\text{TRUE})$

$$\text{w(l)p}(A \,\square\, B,\ R) \equiv \text{w(l)p}(A, R) \wedge \text{w(l)p}(B, R)$$
$$\text{w(l)p}(A\,;B,\ R) \equiv \text{w(l)p}(A,\ \text{w(l)p}(B, R))$$
$$\text{w(l)p}(P \rightarrow A,\ R) \equiv \neg P \vee \text{w(l)p}(A, R)$$
$$\text{w(l)p}(A \boxtimes B,\ R) \equiv \text{w(l)p}(A, R) \wedge (\mathcal{G}(A) \vee \text{w(l)p}(B, R))$$

The equation for $\boxtimes$ follows from those for $\rightarrow$ and $\square$ and the formula

$$A \boxtimes B \equiv A \,\square\, \neg\,\mathcal{G}(A) \rightarrow B\,.$$

The equations for *Fail*, *Skip*, and *Loop* will be left to the reader. For the other three equations, we will first give relational motivations, and then prove that the commands they define satisfy the pairing and conjunctivity conditions.

$\quad\ \text{w(l)p}(A \,\square\, B,\ R)$
$\equiv$ every (proper) outcome of $A \,\square\, B$ satisfies $R$
$\equiv$ every (proper) outcome of $A$ satisfies $R$
$\wedge$ every (proper) outcome of $B$ satisfies $R$
$\equiv \text{w(l)p}(A, R) \wedge \text{w(l)p}(B, R)$

$\quad\ \text{wlp}(A\,;B,\ R)$
$\equiv$ every proper outcome of $A\,;B$ satisfies $R$
$\equiv$ from every proper outcome of $A$, every proper outcome of $B$ satisfies $R$
$\equiv \text{wlp}(A,\ \text{wlp}(B, R))$

$\quad\ \text{wp}(A\,;B,\ R)$
$\equiv$ every outcome of $A\,;B$ satisfies $R$
$\equiv$ every outcome of $A$ is proper and satisfies $\text{wp}(B, R)$
$\equiv \text{wp}(A,\ \text{wp}(B, R))$

$\quad\ \text{w(l)p}(P \rightarrow A,\ R)$
$\equiv$ every (proper) outcome of $P \rightarrow A$ satisfies $R$
$\equiv$ {if $\neg P$ holds then there is no outcome of $P \rightarrow A$}
$\quad\ \neg P \vee$ every (proper) outcome of $A$ satisfies $R$
$\equiv \neg P \vee \text{w(l)p}(A, R)$

The proofs of the pairing and conjunctivity conditions are straightforward calculations. One fact that will be used several times is that in any boolean algebra, finite disjunction distributes over arbitrary conjunctions.

$$
\begin{aligned}
& \mathrm{w(l)p}(A\,;B,\ (\wedge\ R :: R)) \\
\equiv\ & \mathrm{w(l)p}(A,\ \mathrm{w(l)p}(B,\ (\wedge\ R :: R))) \\
\equiv\ & \mathrm{w(l)p}(A,\ (\wedge\ R :: \mathrm{w(l)p}(B, R))) \\
\equiv\ & (\wedge\ R :: \mathrm{w(l)p}(A,\ \mathrm{w(l)p}(B, R))) \\
\equiv\ & (\wedge\ R :: \mathrm{w(l)p}(A\,;B,\ R))
\end{aligned}
$$

$$
\begin{aligned}
& \mathrm{w(l)p}(P \rightarrow A,\ (\wedge\ R :: R)) \\
\equiv\ & \neg P \vee \mathrm{w(l)p}(A,\ (\wedge\ R :: R)) \\
\equiv\ & \neg P \vee (\wedge\ R :: \mathrm{w(l)p}(A, R)) \\
\equiv\ & (\wedge\ R :: \neg P \vee \mathrm{w(l)p}(A, R)) \\
\equiv\ & (\wedge\ R :: \mathrm{w(l)p}(P \rightarrow A,\ R))
\end{aligned}
$$

$$
\begin{aligned}
& \mathrm{w(l)p}(A \,\square\, B,\ (\wedge\ R :: R)) \\
\equiv\ & \mathrm{w(l)p}(A,\ (\wedge\ R :: R)) \wedge \mathrm{w(l)p}(B,\ (\wedge\ R :: R)) \\
\equiv\ & (\wedge\ R :: \mathrm{w(l)p}(A, R)) \wedge (\wedge\ R :: \mathrm{w(l)p}(B, R)) \\
\equiv\ & (\wedge\ R :: \mathrm{w(l)p}(A, R) \wedge \mathrm{w(l)p}(B, R)) \\
\equiv\ & (\wedge\ R :: \mathrm{w(l)p}(A \,\square\, B,\ R))
\end{aligned}
$$

$$
\begin{aligned}
& \mathrm{wp}(A\,;B,\ R) \\
\equiv\ & \mathrm{wp}(A,\ \mathrm{wp}(B, R)) \\
\equiv\ & \mathrm{wp}(A,\ \mathrm{wp}(B, \textsc{true}) \wedge \mathrm{wlp}(B, R)) \\
\equiv\ & \mathrm{wp}(A,\ \mathrm{wp}(B, \textsc{true})) \wedge \mathrm{wp}(A,\ \mathrm{wlp}(B, R)) \\
\equiv\ & \mathrm{wp}(A,\ \mathrm{wp}(B, \textsc{true})) \wedge \mathrm{wp}(A, \textsc{true}) \wedge \mathrm{wlp}(A,\ \mathrm{wlp}(B, R)) \\
\equiv\ & \mathrm{wp}(A,\ \mathrm{wp}(B, \textsc{true})) \wedge \mathrm{wlp}(A,\ \mathrm{wlp}(B, R)) \\
\equiv\ & \mathrm{wp}(A\,;B,\ \textsc{true}) \wedge \mathrm{wlp}(A\,;B,\ R)
\end{aligned}
$$

$$
\begin{aligned}
& \mathrm{wp}(A \,\square\, B,\ R) \\
\equiv\ & \mathrm{wp}(A, R) \wedge \mathrm{wp}(B, R) \\
\equiv\ & \mathrm{wp}(A, \textsc{true}) \wedge \mathrm{wlp}(A, R) \wedge \mathrm{wp}(B, \textsc{true}) \wedge \mathrm{wlp}(B, R) \\
\equiv\ & \mathrm{wp}(A \,\square\, B,\ \textsc{true}) \wedge \mathrm{wlp}(A \,\square\, B,\ R)
\end{aligned}
$$

$$
\begin{aligned}
& \mathrm{wp}(P \rightarrow A,\ R) \\
\equiv\ & \neg P \vee \mathrm{wp}(A, R) \\
\equiv\ & \neg P \vee (\mathrm{wp}(A, \textsc{true}) \wedge \mathrm{wlp}(A, R))
\end{aligned}
$$

$$\equiv (\neg P \vee \mathrm{wp}(A, \text{TRUE})) \wedge (\neg P \vee \mathrm{wlp}(A, R))$$
$$\equiv \mathrm{wp}(P \to A,\ \text{TRUE}) \wedge \mathrm{wlp}(P \to A,\ R)$$

From the precondition equations we derive the following guard equations, using the identity $\mathcal{G}(A) \equiv \neg\,\mathrm{wp}(A, \text{FALSE})$:

$$\mathcal{G}(\mathit{Fail}) \equiv \text{FALSE}$$
$$\mathcal{G}(\mathit{Loop}) \equiv \text{TRUE}$$
$$\mathcal{G}(\mathit{Skip}) \equiv \text{TRUE}$$

$$\mathcal{G}(P \to A) \equiv P \wedge \mathcal{G}(A)$$
$$\mathcal{G}(A \,\Box\, B) \equiv \mathcal{G}(A) \vee \mathcal{G}(B)$$
$$\mathcal{G}(A \boxtimes B) \equiv \mathcal{G}(A) \vee \mathcal{G}(B)$$
$$\mathcal{G}(A\,;B) \equiv \neg\,\mathrm{wp}(A,\ \neg\,\mathcal{G}(B))$$

Note that the last equation implies that $\mathcal{G}(A\ ;\ B) \equiv \mathcal{G}(A)$ whenever $B$ is total.

An important consequence of the conjunctivity of wp and wlp is monotonicity. The proof is very similar to the proof that chain-continuity implies monotonicity:

$P \Rightarrow Q$

$\equiv$ {connection between $\Rightarrow$ and $\wedge$}

$P \wedge Q \equiv P$

$\Rightarrow$ {equal postconditions have equal preconditions}

$\mathrm{w(l)p}(A,\ P \wedge Q) \equiv \mathrm{w(l)p}(A,\ P)$

$\equiv$ {w(l)p is conjunctive}

$\mathrm{w(l)p}(A, P) \wedge \mathrm{w(l)p}(A, Q) \equiv \mathrm{w(l)p}(A, P)$

$\equiv$ {connection between $\wedge$ and $\Rightarrow$}

$\mathrm{w(l)p}(A, P) \Rightarrow \mathrm{w(l)p}(A, Q)$ ∎

## 6 Variables

This section describes two new operations, assignment and projection. To handle them we will rely on the cylindrical properties of the predicates, as well as the boolean properties necessary for the fundamental operations.

If $x$ and $y$ are variables of the same type, the assignment $x := y$ is a command that relates each state $s$ to the state $s'$, where $s'$ coincides with $s$

in all coordinates except $x$, where it has the value $y(s)$. The corresponding predicate transformer is substitution:

$$\mathrm{w(l)p}(x := y,\ R) \equiv R(x : y)\,.$$

Here is the motivation for the equation:

$$\begin{aligned}
& \mathrm{w(l)p}(x := y,\ R) \\
\equiv\ & \text{every (proper) outcome of } x := y \text{ satisfies } R \\
\equiv\ & \text{what } R \text{ says about } x \text{ is true of } y \\
\equiv\ & R(x : y)
\end{aligned}$$

Here is the proof that assignments satisfy the pairing condition:

$$\begin{aligned}
& \mathrm{wp}(x := y,\ \mathrm{TRUE}) \wedge \mathrm{wlp}(x := y,\ R) \\
\equiv\ & \mathrm{TRUE} \wedge R(x : y) \\
\equiv\ & \mathrm{wp}(x := y,\ R)
\end{aligned}$$

The fact that assignment satisfies the conjunctivity condition follows from the fact that in any predicate algebra, substitution distributes over unbounded conjunctions. To prove this, we need a lemma.

**Lemma 1.** *For any predicate $R$ and variables $x$ and $y$,*

$$R(x : y) \;\equiv\; (\forall x : x \neq y \vee R)\,.$$

*Proof.* By the substitutivity of equality,

$$R(x : y) \;\Rightarrow\; x \neq u \vee R\,.$$

The left side is independent of $x$, hence, by the definition of $\forall$, the implication can be strengthened to

$$R(x : y) \;\Rightarrow\; (\forall x : x \neq y \vee R)\,.$$

The proof of the converse is:

$$\begin{aligned}
& (\forall x : x \neq y \vee R) \\
\Rightarrow\ & \{\text{substitutivity of equality}\} \\
& (\forall x : x \neq y \vee R(x : y)) \\
\equiv\ & \{R(x : y) \text{ is independent of } x\} \\
& (\forall x : x \neq y) \vee R(x : y) \\
\equiv\ & \{\text{reflexivity of equality}\} \\
& R(x : y) \quad \blacksquare
\end{aligned}$$

Here is the proof that substitution distributes over arbitrary conjunctions:

$$(\wedge R :: R)(x : y)$$
$$\equiv \{\text{Lemma 1}\}$$
$$(\forall x : x \neq y \vee (\wedge R :: R))$$
$$\equiv \{\text{predicate calculus}\}$$
$$(\wedge R :: (\forall x : x \neq y \vee R))$$
$$\equiv \{\text{Lemma 1}\}$$
$$(\wedge R :: R(x : y))$$

So much for assignment. Next we consider the use of local variables, which requires laying some groundwork.

To indicate that $R$ is independent of all variables except $x$, we will say $R$ is an $x$-predicate. Similarly, an $xy$-predicate is one independent of all variables except $x$ and $y$.

Note that the number of variables on which a predicate depends will vary with the choice of coordinates on the state space. If $R$ is an $xy$-predicate, it is not "two dimensional" in any fundamental way: had the two components $x$ and $y$ been treated as a single component $z$ (with subcomponents $z.x$ and $z.y$), then $R$ would be a $z$-predicate. We will therefore feel free to take all the variables on which a predicate depends and lump them together into a single variable, whenever this is convenient.

A command is called independent of a variable if it neither tests nor sets the variable. In the next section we will make this definition precise, but it will do for now. If $A$ is independent of all variables except $x$, it is called an $x$-command. An $xy$-command or $xyz$-command is defined similarly.

The operation for delimiting the scopes of variables is projection: if $A$ is an $xy$-command, then by $[\![ x \mid A ]\!]$ we denote the $y$-command resulting by dropping the $x$ components from all state-outcome pairs of $A$. (The looping outcome projects to itself.) Operationally, $[\![ x \mid A ]\!]$ means "extend the state space with a new component named $x$ with arbitrary initial value, then execute $A$, then retract the new component". The precondition equation is

$$\mathrm{w(l)p}([\![ x \mid A ]\!], R) \equiv (\forall x : \ \mathrm{w(l)p}(A, R)) \,.$$

Note that if $R$ is a $y$-predicate, then as far as $R$'s value is concerned, we can specify a state by the value of the variable $y$. We will use this notational liberty in the following motivation of the precondition equation:

$\text{w(l)p}(\llbracket x \mid A \rrbracket, R)(y)$
$\equiv$ every (proper) outcome of $\llbracket x \mid A \rrbracket$ from initial state $y$ satisfies $R$
$\equiv$ for every initial value of $x$, every (proper) outcome of $A$ from initial state $(x, y)$ satisfies $R$
$\equiv (\forall x:\ \text{w(l)p}(A, R)(y))$
$\equiv \{y\text{-substitution commutes with } x\text{-quantification}\}$
$(\forall x:\ \text{w(l)p}(A, R))(y)$

The fact that projection satisfies the conjunctivity condition follows from the fact that universal quantification commutes with unbounded conjunction. Here is the proof that projection satisfies the pairing condition:

$$
\begin{aligned}
& \text{wp}(\llbracket x \mid A \rrbracket, R) \\
\equiv\ & (\forall x : \text{wp}(A, R)) \\
\equiv\ & (\forall x : \text{wp}(A, \text{TRUE}) \wedge \text{wlp}(A, R)) \\
\equiv\ & (\forall x : \text{wp}(A, \text{TRUE})) \wedge (\forall x : \text{wlp}(A, R)) \\
\equiv\ & \text{wp}(\llbracket x \mid A \rrbracket, \text{TRUE}) \wedge \text{wlp}(\llbracket x \mid A \rrbracket, R)
\end{aligned}
$$

In the expression $\llbracket x \mid A \rrbracket$, the variable $x$ is a dummy. In case the postcondition is dependent on $x$, the dummy must be renamed to avoid capture before the precondition equation can be used. For example:

$$
\begin{aligned}
& \text{wp}(\llbracket t \mid t := x\,;x := y\,;y := t \rrbracket,\ x < t) \\
\equiv\ & \text{wp}(\llbracket u \mid u := x\,;x := y\,;y := u \rrbracket,\ x < t) \\
\equiv\ & (\forall u : \text{wp}(u := x\,;x := y\,;y := u,\ x < t)) \\
\equiv\ & (\forall u : (x < t)(y : u)(x : y)(u : x)) \\
\equiv\ & (\forall u : y < t) \\
\equiv\ & y < t
\end{aligned}
$$

A common construction in programs has the form

**if** $(\exists x : P)$ **then** let $x$ satisfy $P\,; \ldots$ **end**.

The repetition of $x$ and $P$ are awkward. One longs to write something like

**if** $x$ such that $P$ **then** ... **end**,

where the scope of $x$ includes the ... as well as the $P$. Dijkstra calls such constructions "initializing guards"; they are easily constructed out of partial commands and the projection operator. Note first that

$$\begin{aligned} & \mathcal{G}([\![ x \mid A ]\!]) \\ \equiv\ & \neg\,(\mathrm{wp}([\![ x \mid A ]\!], \text{FALSE})) \\ \equiv\ & \neg\,(\forall x : \mathrm{wp}(A, \text{FALSE})) \\ \equiv\ & \neg\,(\forall x : \neg\,\mathcal{G}(A)) \\ \equiv\ & (\exists x : \mathcal{G}(A)) \end{aligned}$$

Thus if $A$ is total, then the guard of

$$[\![\ x \mid P \to A\ ]\!]$$

is $(\exists x : P)$, and consequently it can be activated in the same states as

$$(\exists x : P) \to A\,,$$

but in the first case the scope of $x$ includes $A$. Thus we have constructed an initializing guard.

As a more concrete example, here is a command that decomposes the list $a$ and rebuilds it in reverse order in $b$, where lists are represented as nil-terminated nests of ordered pairs in the usual way:

$$b := \mathrm{nil}\ ;\ \mathbf{do}\ [\![\ u, v \mid a = (u, v) \to b := (u, b)\ ;\ a := v\ ]\!]\ \mathbf{od}$$

The guard on the loop tests if $a$ is an ordered pair and, if it is, introduces $u$ and $v$ as names for its components. (It would show more respect for the virtue of infix operators to simply write $x \mid A$ instead of $[\![ x \mid A ]\!]$, and dropping the brackets is especially tempting in the case of initializing guards. But it is conventional in programming to delimit name scopes with explicit brackets.)

Finally, a few words about undefined terms. An assignment operation $x := E$, where $E$ is an expression, is considered to be syntactic sugar for

$$\mathbf{if}\ [\![\ u \mid u = E \to x := u\ ]\!]\ \mathbf{fi}\,,$$

where $u$ is a fresh variable. Note that this command loops in states where $E$ is undefined, since in such states $(\exists u : u = E)$ is FALSE. We will abbreviate this formula to $\mathrm{def}(E)$.

In predicate algebras the substitution $(x : E)$, where $E$ is an expression, is defined by the equation

$$P(x : E) \equiv (\exists u :\ u = E \wedge P(x : u))\,,$$

where $u$ is a fresh variable. With this definition we have in general that

$$\text{wp}(x := E,\ R) \equiv R(x : E)$$
$$\text{wlp}(x := E,\ R) \equiv \neg\,\text{def}(E) \vee R(x : E)$$

Note that these are not definitions, but theorems that follow from the semantics of substitution, projection, and **if-fi**. In proofs by structural induction, expressions can generally be ignored.

(A note about the introduction of function symbols into abstract predicate algebras: a function $f$ is defined by giving the meaning of the equation "$u = f(u_1, \ldots u_n)$" as a predicate. For any values of the $v$'s, there must be at most one value of $u$ satisfying the predicate. For example, division can be defined by

$$u = v/w \;\equiv\; w \neq 0 \wedge u * w = v\,.$$

Therefore, if $E$ is not a variable, a predicate expression of the form $u = E$ can be reduced to an expression in which the outer function symbol has been eliminated.)

A final note: by means of the projection operator, predicate transformers can be introduced that are not $\vee$-continuous. That is, $\text{wp}([\![\, x \mid A \,]\!], ?)$ may not distribute over infinite joins of chains of predicates, even though $\text{wp}(A, ?)$ does. This discontinuity occurs for commands that are *unboundedly nondeterministic*, in the sense that they relate some input state to infinitely many outcomes. For example, let $x$ and $y$ have the same type, and consider $[\![\, x \mid y := x \,]\!]$, which assigns $y$ an arbitrary value. For a $y$-predicate $P$,

$$\begin{aligned} &\text{wp}([\![\, x \mid y := x \,]\!], P) \\ \equiv\; &(\forall x : P(y : x)) \\ \equiv\; &[P] \end{aligned}$$

(Recall Dijkstra's [ ] operator from section 4.) The [ ] operator is obviously $\vee$-discontinuous, since the join of a chain of predicates may be TRUE, although none of them is TRUE. But unbounded nondeterminism will not trouble us: although $\text{wp}([\![\, x \mid A \,]\!], R)$ may be a discontinuous function of $R$, we will show that $[\![\, x \mid A \,]\!]$ is a continuous function of $A$, which justifies the use of projection in recursive definitions.

## 7 Commands as predicates

Our next goal is the cograph correspondence. As mentioned in the introduction, the correspondence came out of the predicative programming approach taken by Hehner and Hoare. In this approach, a command is a predicate on a state space that includes variables representing both the initial and final

states of the computation. More precisely, if $A$ is an $x$-command, and $x'$ is a variable of the same type as $x$, then the correspondence between this point of view and the operational view is given by

$A$ as a predicate holds of $(x, x')$
$\equiv$ $x'$ is a possible proper outcome of $A$ as a command activated from initial state $x$.

For example, the command $x := x+1$ corresponds to the predicate $x' = x+1$. Looping outcomes are simply ignored in the usual expositions of this point of view. Our strategy will be to ignore the looping outcomes temporarily, and then recover them by relying on the pairing condition. Instead of $(x, x')$, the initial and final values are usually named $(\grave{x}, \acute{x})$, but we will stick to $(x, x')$, since this simplifies the description of the connection between predicative programming and predicate transformers. This connection is captured exactly by the following theorem.

**Theorem 1.** *If $A$ is an $x$-command, then for any predicate $P$,*

$$\text{wlp}(A, P) \equiv (\forall x' : \text{wlp}(A, x \neq x') \vee P(x : x'))\,.$$

The English translation of the theorem is

$A$'s proper outcomes all satisfy $P$
$\equiv$ for each state $x'$, either $A$ never produces $x'$, or $x'$ satisfies $P$.

Hence the $x$-predicate transformer $\text{wlp}(A, ?)$ is completely determined by the $xx'$-predicate $\text{wlp}(A,\ x \neq x')$. For example, $\text{wlp}(x := x+1,\ ?)$ is determined by $\text{wlp}(x := x+1,\ x \neq x')$, which is $x' \neq x+1$.

The predicate $\text{wlp}(A,\ x \neq x')$ is simply the complement of the predicate that represents $A$ in the predicative programming point of view. Theorem 1 is essentially the work of the Eindhoven Tuesday Afternoon Club [4]; I have modified their theorem to deal with wlp instead of wp, and modified their proof to be axiomatic. To prepare the way for the axiomatic proof, we need one definition and two lemmas.

A command $A$ is *independent* of a variable $x$ if

($i$) for any predicate $R$, $\text{w(l)p}(A, R)$ is independent of $x$

($ii$) for any $x$-predicate $R$, $\text{wlp}(A, R) \equiv \text{wlp}(A, \text{FALSE}) \vee R$.

Condition ($i$) implies that $A$ doesn't test $x$, and condition ($ii$) implies that $A$ doesn't set $x$. We will motivate these claims by arguing in terms of

relations on the state space of $(x, y)$ pairs. First we argue that condition $(i)$ implies that for any values $x_1$, $x_2$, and $y_1$ and any outcome $o$,

$$A(x_1, y_1)o \equiv A(x_2, y_1)o\,,$$

*i.e.*, the outcome is indifferent to the initial value of $x$. The proper and improper cases are treated separately:

$$\begin{array}{ll} & A(x_1, y_1)\bot \\ \equiv & \neg\, \mathrm{wp}(A, \mathrm{TRUE})(x_1, y_1) \\ \equiv & \{(i)\} \\ & \neg\, \mathrm{wp}(A, \mathrm{TRUE})(x_2, y_1) \\ \equiv & A(x_2, y_1)\bot \end{array}$$

$$\begin{array}{ll} & A(x_1, y_1)o \\ \equiv & \{\text{assume } o \text{ proper}\} \\ & \neg\, \mathrm{wp}(A, \overline{o})(x_1, y_1) \\ \equiv & \{(i)\} \\ & \neg\, \mathrm{wp}(A, \overline{o})(x_2, y_1) \\ \equiv & A(x_2, y_1)o \end{array}$$

Next we argue that condition $(ii)$ implies that for any $x_1$, $x_2$, $y_1$, and $y_2$,

$$A(x_1, y_1)(x_2, y_2) \;\Rightarrow\; x_1 = x_2\,,$$

*i.e.*, in no outcome is $x$ changed:

$$\begin{array}{ll} & A(x_1, y_1)(x_2, y_2) \\ \equiv & \neg\, \mathrm{wlp}(A,\ \overline{(x_2, y_2)})(x_1, y_1) \\ \equiv & \neg\, \mathrm{wlp}(A,\ x \neq x_2 \vee y \neq y_2)(x_1, y_1) \\ \Rightarrow & \{\text{wlp is monotonic, } \neg \text{ is anti-montonic}\} \\ & \neg\, \mathrm{wlp}(A, x \neq x_2)(x_1, y_1) \\ \Rightarrow & \{(ii),\ x \neq x_2 \text{ is an } x\text{-predicate}\} \\ & x_1 = x_2 \end{array}$$

So much for the motivation. A command and predicate are said to be *independent* if there is no variable on which both are dependent. Note that in this case, clause $(ii)$ of the definition above applies to them.

**Lemma 2.** *If command A and predicate C are independent, then*

$$\text{wlp}(A,\ P \lor C) \equiv \text{wlp}(A, P) \lor C\,.$$

*Proof.* Since wlp is not disjunctive, it is not obvious how to get the proof started. There is a roundabout attack that succeeds in this case, as in many similar ones. Compute first that

$$
\begin{array}{ll}
 & \text{TRUE} \\
\equiv & \text{wlp}(A, \text{FALSE}) \lor C \lor \neg C \\
\equiv & \{A \text{ and } C \text{ are independent}\} \\
 & \text{wlp}(A, C) \lor \text{wlp}(A, \neg C) \\
\Rightarrow & \{\text{wlp is monotonic}\} \\
 & \text{wlp}(A, C) \lor \text{wlp}(A,\ P \lor \neg C)
\end{array}
$$

and then that

$$
\begin{array}{ll}
 & \text{wlp}(A,\ P \lor C) \\
\equiv & \text{wlp}(A,\ P \lor C) \land (\text{wlp}(A, C) \lor \text{wlp}(A,\ P \lor \neg C) \\
\equiv & \{\ \land \text{ over } \lor;\ \text{conjunctivity of wlp}\} \\
 & \text{wlp}(A, C) \lor \text{wlp}(A, P) \\
\equiv & \{A \text{ and } C \text{ are independent}\} \\
 & \text{wlp}(A, \text{FALSE}) \lor C \lor \text{wlp}(A, P) \\
\equiv & \{\text{monotonicity of wlp}\} \\
 & C \lor \text{wlp}(A, P) \quad \blacksquare
\end{array}
$$

**Lemma 3.** *If predicate P is independent of $x'$, then*

$$P \equiv (\forall x' : x' \neq x \lor P(x : x'))\,.$$

*Proof.* This is a trivial consequence of Lemma 1:

$$
\begin{array}{ll}
 & (\forall x' : x' \neq x \lor P(x : x')) \\
\equiv & \{\text{Lemma 1 with } R := P(x : x')\} \\
 & P(x : x')(x' : x) \\
\equiv & \{P \text{ is independent of } x'\} \\
 & P \quad \blacksquare
\end{array}
$$

Finally we are ready for the proof of Theorem 1, which is now very simple:

$$\begin{aligned}
& \mathrm{wlp}(A, P) \\
\equiv\; & \{\text{Lemma 3}\} \\
& \mathrm{wlp}(A,\ (\forall x' : x \neq x' \vee P(x : x'))) \\
\equiv\; & \{\text{conjunctivity of wlp}\} \\
& (\forall x' : \mathrm{wlp}(A,\ x \neq x' \vee P(x : x'))) \\
\equiv\; & \{\text{Lemma 2}\} \\
& (\forall x' : \mathrm{wlp}(A,\ x \neq x') \vee P(x : x')) \quad \blacksquare
\end{aligned}$$

Given Theorem 1, the construction of the cograph correspondence is routine:

**Definitions.** For an $x$-command $A$, define

$$\begin{aligned}
H(A) &= \mathrm{wp}(A, \textsc{true}) \\
G(A) &= \mathrm{wlp}(A, x \neq x') \\
\mathrm{cog}(A) &\equiv (H(A), G(A))
\end{aligned}$$

Note that $G(A)$ holds of $(x, x')$ if $x'$ is not a possible outcome of $A$ from $x$, and $H(A)$ holds of $x$ if $\perp$ is not a possible outcome of $A$ from $x$, so that $\mathrm{cog}(A)$ (read "cograph of $A$") represents the complement of the graph of $A$ as a relation.

Inversely, if $H$ is an $x$-predicate and $G$ is an $xx'$-predicate, define the command $\mathrm{com}(H, G)$ by the rules

$$\begin{aligned}
\mathrm{wlp}(\mathrm{com}(H,G),\ R) &\equiv (\forall x' : G \vee R(x : x')) \\
\mathrm{wp}(\mathrm{com}(H,G),\ R) &\equiv H \wedge \mathrm{wlp}(\mathrm{com}(H,G),\ R)
\end{aligned}$$

Obviously $\mathrm{com}(H, G)$ satisfies the pairing constraint. The fact that it also satisfies the wlp conjunctivity constraint follows from the fact that substitution, finite disjunction, and universal quantification all distribute over arbitrary conjunctions. For the wp conjunctivity constraint, one uses in addition the fact that finite conjunction distributes over arbitrary non-empty conjunctions.

**Theorem 2.** *The operators cog and com are inverses.*

*Proof.*

$$\begin{aligned}
& H(\mathrm{com}(H, G)) \\
\equiv\; & \{\text{definition of } H\}
\end{aligned}$$

$$\begin{aligned}
&\quad \mathrm{wp}(\mathrm{com}(H,G),\ \mathrm{TRUE}) \\
&\equiv \{\text{definition of com}\} \\
&\quad H
\end{aligned}$$

$$\begin{aligned}
&\quad G(\mathrm{com}(H,G)) \\
&\equiv \{\text{definition of } G\} \\
&\quad \mathrm{wlp}(\mathrm{com}(H,G),\ x \neq x') \\
&\equiv \{\text{definition of com, use } x'' \text{ instead of } x' \text{ to avoid capture}\} \\
&\quad (\forall x'' : G(x' : x'') \vee (x \neq x')(x : x'')) \\
&\equiv (\forall x'' : G(x' : x'') \vee x'' \neq x') \\
&\equiv \{\text{Lemma 3}\} \\
&\quad G
\end{aligned}$$

Hence $\mathrm{cog}(\mathrm{com}(H,G)) = (H,G)$. To show $\mathrm{com}(\mathrm{cog}(A))$ is $A$, we will show they have the same wlp's for arbitrary postconditions, and the same wp's for TRUE.

$$\begin{aligned}
&\quad \mathrm{wp}(\mathrm{com}(\mathrm{cog}(A)),\ \mathrm{TRUE}) \\
&\equiv \{\text{definition of cog}\} \\
&\quad \mathrm{wp}(\mathrm{com}(H(A), G(A)),\ \mathrm{TRUE}) \\
&\equiv \{\text{definition of com}\} \\
&\quad H(A) \\
&\equiv \{\text{definition of } H\} \\
&\quad \mathrm{wp}(A, \mathrm{TRUE})
\end{aligned}$$

$$\begin{aligned}
&\quad \mathrm{wlp}(\mathrm{com}(\mathrm{cog}(A)),\ R) \\
&\equiv \{\text{definition of cog}\} \\
&\quad \mathrm{wlp}(\mathrm{com}(H(A), G(A)),\ R) \\
&\equiv \{\text{definition of com and of } G\} \\
&\quad (\forall x' : \mathrm{wlp}(A,\ x \neq x') \vee R(x : x')) \\
&\equiv \{\text{Theorem 1}\} \\
&\quad \mathrm{wlp}(A, R) \quad \blacksquare
\end{aligned}$$

As a further illustration of the cograph correspondence, Table 2 lists the commands whose cographs contain only the simple predicates TRUE, FALSE, and $x \neq x'$. Our old friends *Fail*, *Skip*, and *Loop* are present, and also a

new command *Havoc*, which was not mentioned before because it is not very useful. It relates each initial state to every proper outcome. Its precondition equation is

$$\mathrm{w(l)p}(\mathit{Havoc}, P) \equiv [P]\,.$$

In the HH model that is usually associated with the programs-as-predicates approach, *Havoc* can be described (it corresponds to the $xx'$-predicate TRUE), but *Loop* cannot. Because of this, Hehner [11] bestows upon *Havoc* by fiat properties that *Loop* enjoys naturally. For example, he defines semicolon so that $\mathit{Havoc}\,;\, x := 0 \equiv \mathit{Havoc}$. With the ordinary semicolon, we would have $\mathit{Havoc}\,;\, x := 0 \equiv x := 0$ (assuming we are talking about $x$-commands), while of course $\mathit{Loop}; x := 0 \equiv \mathit{Loop}$. One consequence of this is that the semicolon becomes non-associative. This can be fixed by adding a distinguished variable to represent termination; but then the HH model reduces to the general model, with the distinguished variable playing a role similar to $\perp$.

| $A$ | $H(A)$ | $G(A)$ |
|---|---|---|
| *Fail* | TRUE | TRUE |
| *Loop* | FALSE | TRUE |
| *Havoc* | TRUE | FALSE |
| *Loop* $\square$ *Havoc* | FALSE | FALSE |
| *Skip* | TRUE | $x \neq x'$ |
| *Skip* $\square$ *Loop* | FALSE | $x \neq x'$ |

**Table 2.** Some commands with simple cographs.

## 8 Recursion

Finally we come to the problem with which this research started: to apply the cograph correspondence and the fixpoint method to general recursion in the calculus of guarded commands. To handle recursion we will rely on the completeness of the predicate algebra, in addition to the boolean and cylindrical properties we have relied on up till now.

Here is an operational description of the fixpoint method. Let $B$ be some command, and for each $n$ let $A_n$ be the command "execute $B$ for at most $n$ steps, abandoning any longer computations and classifying them as loops". It should be clear that in some sense, each $A_n$ is an approximation to $B$, and that the approximation improves as $n$ increases. The key to the fixpoint method is to define this notion of approximation without reference to computation sequences.

This is straightforward enough. We consider a relational definition first, then switch to predicate transformers. In the relational model, we will say that $A$ approximates $B$ if $A$ can be obtained from $B$ by replacing some of $B$'s proper outcomes with the looping outcome. This condition is equivalent to the following two constraints, in which $s$ and $t$ range over states and $o$ ranges over outcomes:

(*i*) $(\forall s,t : Ast \Rightarrow Bst)$

(*ii*) $(\forall s,o : Bso \Rightarrow (Aso \vee As\bot))$

Condition (*i*) says that every proper outcome of the approximation $A$ is a proper outcome of the limit $B$. Condition (*ii*) says that every outcome of the limit is either an outcome of the approximation, or is represented in the approximation by the looping outcome.

Translating into the language of predicate transformers, we have from (*i*) (again using the notation $\bar{s}$ for the co-atomic predicate that holds for all states except $s$)

$$
\begin{aligned}
& (\forall s,t : Ast \Rightarrow Bst) \\
\equiv\ & (\forall s,t : \neg\,\mathrm{wlp}(A,\bar{t})(s) \Rightarrow \neg\,\mathrm{wlp}(B,\bar{t})(s)) \\
\equiv\ & (\forall R : R \text{ co-atomic} : \mathrm{wlp}(B,R) \Rightarrow \mathrm{wlp}(A,R)) \\
\equiv\ & \{\text{every predicate is a conjunction of co-atoms, conjunctivity of wlp}\} \\
& (\forall R : R \text{ a predicate} : \mathrm{wlp}(B,R) \Rightarrow \mathrm{wlp}(A,R))
\end{aligned}
$$

and from (*ii*)

$$
\begin{aligned}
& (\forall s,o : Bso \Rightarrow (Aso \vee As\bot)) \\
\equiv\ & (\forall s : \neg\,As\bot \Rightarrow (\forall o : Bso \Rightarrow Aso)) \\
\equiv\ & (\forall s : \neg\,As\bot \Rightarrow (\neg\,Bs\bot \wedge (\forall t : Bst \Rightarrow Ast))) \\
\equiv\ & \mathrm{wp}(A,\textsc{true}) \Rightarrow \\
& \qquad \mathrm{wp}(B,\textsc{true}) \wedge (\forall R : R \text{ a predicate} : \mathrm{wlp}(A,R) \Rightarrow \mathrm{wlp}(B,R)) \\
\equiv\ & (\forall R : R \text{ a predicate} : \mathrm{wp}(A,R) \Rightarrow \mathrm{wp}(B,R))
\end{aligned}
$$

Thus we have derived the pleasantly symmetrical definition

$$A \sqsubseteq B \;\equiv\; A \sqsubseteq_{\mathrm{wp}} B \;\wedge\; B \sqsubseteq_{\mathrm{wlp}} A\,,$$

where

$$A \sqsubseteq_{\mathrm{wp}} B \equiv (\forall R : \mathrm{wp}(A,R) \Rightarrow \mathrm{wp}(B,R))$$

and

$$A \sqsubseteq_{\mathrm{wlp}} B \equiv (\forall R : \mathrm{wlp}(A, R) \Rightarrow \mathrm{wlp}(B, R)) \,.$$

The partial order $\sqsubseteq$ is the required approximation relation. Note that both $\sqsubseteq_{\mathrm{wp}}$ and $\sqsubseteq_{\mathrm{wlp}}$ are transitive and reflexive. These properties are preserved under inversion and intersection; hence $\sqsubseteq$, which is the intersection of $\sqsubseteq_{\mathrm{wp}}$ with the inverse of $\sqsubseteq_{\mathrm{wlp}}$, is also reflexive and transitive. Furthermore, if $A \sqsubseteq B$ and $B \sqsubseteq A$, then $A$ and $B$ have the same wp's and wlp's for every predicate; therefore they are the same command. Hence $\sqsubseteq$ is a reflexive partial order.

The next step in the method of fixpoint semantics is to consider the joins of chains in the approximation order. The axiomatic program requires that we base our proof of the existence of joins upon the completeness of the set of predicates. The two requirements are most easily met by using the cograph correspondence. We therefore investigate the approximation relation $\sqsubseteq$ in this representation.

**Definition.** For cographs $(H, G)$ and $(H', G')$, we define $(H, G) \sqsubseteq (H', G')$ to mean

$$(H \Rightarrow H') \wedge (G' \Rightarrow G) \wedge (H \wedge G \Rightarrow G') \,.$$

**Theorem 3.** *For cographs $(H, G)$ and $(H', G')$,*

$$((H, G) \sqsubseteq (H', G')) \equiv (com(H, G) \sqsubseteq com(H', G')) \,.$$

*Proof.* Assume first that $(H, G) \sqsubseteq (H', G')$. Then, first, for any $R$

$$\begin{aligned}
& \mathrm{wp}(\mathrm{com}(H, G),\ R) \\
\equiv\ & \mathrm{wp}(\mathrm{com}(H, G),\ \mathrm{TRUE}) \wedge \mathrm{wlp}(\mathrm{com}(H, G),\ R) \\
\equiv\ & H \wedge (\forall x' : G \vee R(x : x')) \\
\Rightarrow\ & \{H \Rightarrow H',\ H \wedge G \Rightarrow G'\} \\
& H' \wedge (\forall x' : G' \vee R(x : x')) \\
\equiv\ & \mathrm{wp}(\mathrm{com}(H', G'),\ R)
\end{aligned}$$

and, second, for any $R$

$$\begin{aligned}
& \mathrm{wlp}(\mathrm{com}(H', G'),\ R) \\
\equiv\ & (\forall x' : G' \vee R(x : x')) \\
\Rightarrow\ & \{G' \Rightarrow G\}
\end{aligned}$$

$$
\begin{aligned}
&\quad (\forall x' : G \vee R(x : x')) \\
&\equiv \text{wlp}(\text{com}(H, G),\ R)
\end{aligned}
$$

and therefore $\text{com}(H, G) \sqsubseteq \text{com}(H', G')$.

To prove the converse, let $A$ and $B$ be commands such that $A \sqsubseteq B$. Then first,

$$
\begin{aligned}
&\quad H(A) \\
&\equiv \text{wp}(A, \text{TRUE}) \\
&\Rightarrow \{A \sqsubseteq_{\text{wp}} B\} \\
&\quad \text{wp}(B, \text{TRUE}) \\
&\equiv H(B)
\end{aligned}
$$

and second,

$$
\begin{aligned}
&\quad G(B) \\
&\equiv \text{wlp}(B,\ x \neq x') \\
&\Rightarrow \{B \sqsubseteq_{\text{wlp}} A\} \\
&\quad \text{wlp}(A,\ x \neq x') \\
&\equiv G(A)
\end{aligned}
$$

and third,

$$
\begin{aligned}
&\quad H(A) \wedge G(A) \\
&\equiv \text{wp}(A, \text{TRUE}) \wedge \text{wlp}(A,\ x \neq x') \\
&\equiv \{\text{pairing condition}\} \\
&\quad \text{wp}(A,\ x \neq x') \\
&\Rightarrow \{A \sqsubseteq_{\text{wp}} B\} \\
&\quad \text{wp}(B,\ x \neq x') \\
&\Rightarrow \{\text{pairing condition}\} \\
&\quad \text{wlp}(B,\ x \neq x') \\
&\equiv G(B)
\end{aligned}
$$

and therefore $(H(A), G(A)) \sqsubseteq (H(B), G(B))$. ∎

From the formula in Theorem 3, we can easily derive and verify a formula for the join of a chain.

**Theorem 4.** *Any chain has a join, which is given in the cograph representation by the formula*

$$(\sqcup\ H, G :: (H, G)) \equiv ((\vee\ H, G :: H),\ (\wedge\ H, G :: G))\,.$$

*Proof.* To show that the given cograph is an upper bound, let $(H, G)$ be any cograph in the chain, and check first that

$$H \Rightarrow (\vee\ H, G :: H)\,,$$

and second that

$$(\wedge\ H, G :: G) \Rightarrow G\,,$$

and third (observing that $H \wedge G \Rightarrow G'$ follows both from $(H, G) \sqsubseteq (H', G')$ and from $(H', G') \sqsubseteq (H, G)$) that

$$\begin{array}{ll} & H \wedge G \Rightarrow (\wedge\ H', G' :: G') \\ \equiv & (\wedge\ H', G' :: H \wedge G \Rightarrow G') \\ \equiv & \{(H,G) \text{ and } (H',G') \text{ are in the chain; observation above}\} \\ & \text{TRUE} \end{array}$$

and therefore the given cograph is an upper bound for the chain. Now let $(H', G')$ be some other upper bound, so that for any $(H, G)$ in the chain,

$(i)$ $H \Rightarrow H'$

$(ii)$ $G' \Rightarrow G$

$(iii)$ $H \wedge G \Rightarrow G'$

We must prove that

$$((\vee\ H, G :: H), (\wedge\ H, G :: G)) \sqsubseteq (H', G')\,,$$

which we do by observing first that

$$\begin{array}{ll} & (\vee\ H, G :: H) \Rightarrow H' \\ \equiv & (\wedge\ H, G :: H \Rightarrow H') \\ \equiv & \{(i)\} \\ & \text{TRUE} \end{array}$$

and second that

$$
\begin{aligned}
& G' \Rightarrow (\wedge\ H, G :: G) \\
\equiv\ & (\wedge\ H, G :: G' \Rightarrow G) \\
\equiv\ & \{(ii)\} \\
& \text{TRUE}
\end{aligned}
$$

and third that

$$
\begin{aligned}
& (\vee\ H, G :: H) \wedge (\wedge\ H, G :: G) \Rightarrow G' \\
\Leftarrow\ & (\vee\ H, G :: H \wedge G) \Rightarrow G' \\
\equiv\ & (\wedge\ H, G :: H \wedge G \Rightarrow G') \\
\equiv\ & \{(iii)\} \\
& \text{TRUE}
\end{aligned}
$$

which completes the proof. ∎

Note that if a command has no looping outcome, then it is a maximal element under the approximation order: the only command it approximates is itself. Thus there are many maximal elements, and therefore no maximum element. Thus the empty set does not have a meet. But, as it turns out, any nonempty set does have a meet. This result will allow us to apply our version of the Knaster-Tarski Theorem when we need it in the proof of the final theorem on recursion.

A note on notation: since we are using $\equiv$ for the equality *relation* on predicates, we use $=$ as the equality *operation*. That is, $P = Q$ means $(\neg P \vee Q) \wedge (P \vee \neg Q)$.

**Theorem 5.** *Let $(H_i, G_i)$ be a non-empty indexed set of cographs, where $i$ ranges over an anonymous index set. Define $H'$ and $G'$ by*

$$
\begin{aligned}
H' &\equiv (\wedge\ i, j :: H_i \wedge (G_i = G_j)) \\
G' &\equiv (\vee\ i :: G_i)
\end{aligned}
$$

*Then $(H', G')$ is the greatest lower bound for the set $(H_i, G_i)$ in the $\sqsubseteq$ order.*

*Proof.* Here is the proof that $(H', G')$ is a lower bound:

$$
\begin{aligned}
& (H', G') \sqsubseteq (H_i, G_i) \\
\equiv\ & (H' \Rightarrow H_i) \wedge (G_i \Rightarrow G') \wedge (H' \wedge G' \Rightarrow G_i) \\
\equiv\ & \{H' \Rightarrow (\wedge\ i :: H_i) \Rightarrow H_i\} \\
& (G_i \Rightarrow G') \wedge (H' \wedge G' \Rightarrow G_i) \\
\equiv\ & \{G_i \Rightarrow (\vee\ i :: G_i) \equiv G'\}
\end{aligned}
$$

$$
\begin{array}{ll}
& H' \wedge G' \Rightarrow G_i \\
\Leftarrow & \{\text{weaken } H'\} \\
& (\wedge\, i,j : G_i = G_j) \wedge (\vee\, i : G_i) \Rightarrow G_i \\
\equiv & \{\text{the } G\text{'s are equal and one is true } \Rightarrow \text{ any is true}\} \\
& \text{TRUE}
\end{array}
$$

To show that $(H', G')$ is the greatest lower bound, let $(H, G)$ be another lower bound; thus for each $i$,

$(i)$ $H \Rightarrow H_i$

$(ii)$ $G_i \Rightarrow G$

$(iii)$ $H \wedge G \Rightarrow G_i$

To show that $(H, G) \sqsubseteq (H', G')$, check first that

$$
\begin{array}{ll}
& H \Rightarrow H' \\
\equiv & H \Rightarrow (\wedge\, i,j :: H_i \wedge (G_i = G_j)) \\
\equiv & (\wedge\, i,j :: H \Rightarrow H_i \wedge (G_i = G_j)) \\
\equiv & \{(i)\} \\
& (\wedge\, i,j :: H \Rightarrow (G_i = G_j)) \\
\equiv & (\wedge\, i,j :: H \wedge G_i \Rightarrow G_j) \\
\Leftarrow & \{(ii)\} \\
& (\wedge\, i,j :: H \wedge G \Rightarrow G_j) \\
\equiv & \{(iii)\} \\
& \text{TRUE}
\end{array}
$$

and second that

$$
\begin{array}{ll}
& G' \Rightarrow G \\
\equiv & (\vee\, i :: G_i) \Rightarrow G \\
\equiv & \{(ii)\} \\
& \text{TRUE}
\end{array}
$$

and finally that

$$
\begin{array}{ll}
& H \wedge G \Rightarrow G' \\
\equiv & H \wedge G \Rightarrow (\vee\, i :: G_i) \\
\Leftarrow & (\vee\, i :: H \wedge G \Rightarrow G_i)
\end{array}
$$

$$\equiv \{(iii)\}$$
$$(\vee\, i :: \text{TRUE})$$
$$\equiv \{i \text{ ranges over a non-empty set}\}$$
$$\text{TRUE}$$

which completes the proof. ∎

Next we consider the distribution properties of wp and wlp over joins of chains.

**Theorem 6.** *Let $A$ range over any chain of commands, then for any predicate $R$,*

$$\text{w(l)p}((\sqcup\, A :: A),\ R) \equiv (\vee(\wedge)\ A :: \text{w(l)p}(A, R))\,.$$

*Proof.* In the following, $(H, G)$ ranges over the chain of cographs corresponding to the chain of commands that $A$ ranges over. Here is the proof of the formula for wlp:

$$\text{wlp}((\sqcup\, A :: A),\ R)$$
$$\equiv \text{wlp}((\sqcup\, H, G :: \text{com}(H, G)),\ R)$$
$$\equiv \{\text{Theorem 4}\}$$
$$\text{wlp}(\text{com}((\vee\, H, G :: H), (\wedge\, H, G :: G)),\ R)$$
$$\equiv \{\text{def. of com}\}$$
$$(\forall x' : (\wedge\, H, G :: G) \vee R(x : x'))$$
$$\equiv (\forall x' : (\wedge\, H, G :: G \vee R(x : x')))$$
$$\equiv (\wedge\, H, G :: (\forall x' : G \vee R(x : x')))$$
$$\equiv \{\text{def. of com}\}$$
$$(\wedge\, H, G :: \text{wlp}(\text{com}(H, G),\ R))$$
$$\equiv (\wedge\, A :: \text{wlp}(A, R))$$

The two directions for the formula for wp are proved separately:

$$\text{wp}((\sqcup\, A :: A),\ R)$$
$$\equiv \text{wp}((\sqcup\, H, G :: \text{com}(H, G)),\ R)$$
$$\equiv \{\text{Theorem 4}\}$$
$$\text{wp}(\text{com}((\vee\, H, G :: H), (\wedge\, H, G :: G)),\ R)$$
$$\equiv \{\text{pairing condition, def. of com}\}$$
$$(\vee\, H, G :: H) \wedge (\forall x' : (\wedge\, H, G :: G) \vee R(x : x'))$$

$$\begin{aligned}
&\equiv (\vee H, G :: H \wedge (\forall x' : (\wedge H, G :: G) \vee R(x : x')))\\
&\Rightarrow (\vee H, G :: H \wedge (\forall x' : G \vee R(x : x')))\\
&\equiv (\vee H, G :: \mathrm{wp}(\mathrm{com}(H, G),\ R))\\
&\equiv (\vee A :: \mathrm{wp}(A, R))
\end{aligned}$$

The proof of the other direction is

$$\begin{aligned}
&(\vee A :: \mathrm{wp}(A, R)) \Rightarrow \mathrm{wp}((\sqcup A :: A),\ R)\\
&\equiv (\wedge A :: \mathrm{wp}(A, R) \Rightarrow \mathrm{wp}((\sqcup A :: A),\ R))\\
&\equiv \{A \sqsubseteq (\sqcup A :: A)\}\\
&\quad \mathrm{TRUE}
\end{aligned}$$

This completes the proof. ∎

**Theorem 7.** *The fundamental operators, assignment, and projection are all chain-continuous in their command arguments.*

*Proof.* In order to express the proof compactly, the parenthesis convention will be used repeatedly. So will the principle that two commands are equal if they have the same w(l)p's for any postcondition.

Here is the proof that $\Box$ is chain-continuous in its first (hence also in its second) argument:

$$\begin{aligned}
&\mathrm{w(l)p}((\sqcup A :: A) \mathbin{\Box} B,\ R)\\
&\equiv \mathrm{w(l)p}((\sqcup A :: A),\ R) \wedge \mathrm{w(l)p}(B, R)\\
&\equiv (\vee(\wedge)\ A :: \mathrm{w(l)p}(A, R)) \wedge \mathrm{w(l)p}(B, R)\\
&\equiv (\vee(\wedge)\ A :: \mathrm{w(l)p}(A, R) \wedge \mathrm{w(l)p}(B, R))\\
&\equiv (\vee(\wedge)\ A :: \mathrm{w(l)p}(A \mathbin{\Box} B, R))\\
&\equiv \mathrm{w(l)p}((\sqcup A :: A \mathbin{\Box} B),\ R)
\end{aligned}$$

The proof that $\rightarrow$ is chain-continuous in its second argument is very similar:

$$\begin{aligned}
&\mathrm{w(l)p}(P \rightarrow (\sqcup A :: A),\ R)\\
&\equiv \neg P \vee \mathrm{w(l)p}((\sqcup A :: A),\ R)\\
&\equiv \neg P \vee (\vee(\wedge)\ A :: \mathrm{w(l)p}(A, R))\\
&\equiv (\vee(\wedge)\ A :: \neg P \vee \mathrm{w(l)p}(A, R))\\
&\equiv (\vee(\wedge)\ A :: \mathrm{w(l)p}(P \rightarrow A, R))\\
&\equiv \mathrm{w(l)p}((\sqcup A :: P \rightarrow A),\ R)
\end{aligned}$$

Here is the proof that the operator ; is chain-continuous in its first argument:

$$
\begin{aligned}
&\mathrm{w(l)p}((\sqcup\ A :: A)\ ;B,\ R)\\
\equiv\ &\mathrm{w(l)p}((\sqcup\ A :: A),\ \mathrm{w(l)p}(B, R))\\
\equiv\ &(\vee(\wedge)\ A :: \mathrm{w(l)p}(A, \mathrm{w(l)p}(B, R)))\\
\equiv\ &(\vee(\wedge)\ A :: \mathrm{w(l)p}(A\ ; B,\ R))\\
\equiv\ &\mathrm{w(l)p}((\sqcup\ A :: A\ ; B),\ R)
\end{aligned}
$$

And in its second argument:

$$
\begin{aligned}
&\mathrm{w(l)p}(A\ ;(\sqcup\ B :: B),\ R)\\
\equiv\ &\mathrm{w(l)p}(A,\ \mathrm{w(l)p}((\sqcup\ B :: B),\ R))\\
\equiv\ &\mathrm{w(l)p}(A,\ (\vee(\wedge)\ B :: \mathrm{w(l)p}(B, R)))\\
\equiv\ &\{\text{explanation below}\}\\
&(\vee(\wedge)\ B :: \mathrm{w(l)p}(A,\ \mathrm{w(l)p}(B, R)))\\
\equiv\ &(\vee(\wedge)\ B :: \mathrm{w(l)p}(A\ ; B,\ R))\\
\equiv\ &\mathrm{w(l)p}((\sqcup\ B :: A\ ; B),\ R)
\end{aligned}
$$

Explanation of the step above: the wlp case is a simple consequence of wlp's conjunctivity. The wp case is less obvious, since wp is not disjunctive in general. But it is easy to show that if $B1 \Rightarrow B2$, then wp distributes over the disjunction $B1 \vee B2$. Since the $B$'s range over a chain, any two values of $\mathrm{wp}(B, R)$ will be comparable by $\Rightarrow$. Therefore the step is also valid in the wp case.

Before considering $\boxtimes$, observe that

$$
\begin{aligned}
&\mathcal{G}((\sqcup\ A :: A))\\
\equiv\ &\neg\mathrm{wp}((\sqcup\ A :: A),\ \textsc{false})\\
\equiv\ &\neg(\vee\ A :: \mathrm{wp}(A, \textsc{false}))\\
\equiv\ &\neg(\vee\ A :: \neg\mathcal{G}(A))\\
\equiv\ &(\wedge\ A :: \mathcal{G}(A))
\end{aligned}
$$

For the first argument to $\boxtimes$, we consider the wp and wlp cases separately. The wlp case is easy:

$$
\begin{aligned}
&\mathrm{wlp}((\sqcup\ A :: A) \boxtimes B,\ R)\\
\equiv\ &\mathrm{wlp}((\sqcup\ A :: A),\ R) \wedge (\mathcal{G}((\sqcup\ A :: A)) \vee \mathrm{wlp}(B, R))\\
\equiv\ &(\wedge\ A :: \mathrm{wlp}(A, R)) \wedge (\wedge\ A :: \mathcal{G}(A) \vee \mathrm{wlp}(B, R))\\
\equiv\ &(\wedge\ A :: \mathrm{wlp}(A, R) \wedge (\mathcal{G}(A) \vee \mathrm{wlp}(B, R)))
\end{aligned}
$$

$$\begin{aligned}
&\equiv (\wedge\ A :: \mathrm{wlp}(A \boxtimes B,\ R)) \\
&\equiv \mathrm{wlp}((\sqcup\ A :: A \boxtimes B),\ R)
\end{aligned}$$

The wp case is harder:

$$\begin{aligned}
&\quad \mathrm{wp}((\sqcup\ A :: A) \boxtimes B,\ R) \\
&\equiv \mathrm{wp}((\sqcup\ A :: A),\ R) \wedge (\mathcal{G}((\sqcup\ A :: A)) \vee \mathrm{wp}(B, R)) \\
&\equiv (\vee\ A :: \mathrm{wp}(A, R)) \wedge ((\wedge\ A :: \mathcal{G}(A)) \vee \mathrm{wp}(B, R)) \\
&\equiv (\vee\ A :: \mathrm{wp}(A, R) \wedge ((\wedge\ A :: \mathcal{G}(A)) \vee \mathrm{wp}(B, R))) \\
&\equiv \{\text{explanation below}\} \\
&\quad (\vee\ A :: \mathrm{wp}(A, R) \wedge (\mathcal{G}(A) \vee \mathrm{wp}(B, R))) \\
&\equiv (\vee\ A :: \mathrm{wp}(A \boxtimes B,\ R)) \\
&\equiv \mathrm{wp}((\sqcup\ A :: A \boxtimes B),\ R)
\end{aligned}$$

Explanation of the step above: To show that $(\wedge\ A :: \mathcal{G}(A))$ may be replaced by $\mathcal{G}(A)$ in the context in which it appears in the step above, we just need to show that for any $A'$ such that $A \sqsubseteq A'$ or $A' \sqsubseteq A$,

$$\mathrm{wp}(A, R) \wedge \mathcal{G}(A) \Rightarrow \mathcal{G}(A') .$$

This follows from two observations. First, that

$$A' \sqsubseteq A \Rightarrow (\mathcal{G}(A) \Rightarrow \mathcal{G}(A')) ,$$

since

$$\mathcal{G}(A) \equiv \neg\,\mathrm{wp}(A, \textsc{false}) \Rightarrow \neg\,\mathrm{wp}(A', \textsc{false}) \equiv \mathcal{G}(A') .$$

Second, that

$$A \sqsubseteq A' \Rightarrow (\mathcal{G}(A) \wedge \mathrm{wp}(A, \textsc{true}) \Rightarrow \mathcal{G}(A')) ,$$

since

$$\begin{aligned}
&\quad \mathcal{G}(A) \wedge \mathrm{wp}(A, \textsc{true}) \\
&\equiv \neg\,\mathrm{wp}(A, \textsc{false}) \wedge \mathrm{wp}(A, \textsc{true}) \\
&\equiv \{\text{pairing condition}\} \\
&\quad \neg\,(\mathrm{wp}(A, \textsc{true}) \wedge \mathrm{wlp}(A, \textsc{false})) \wedge \mathrm{wp}(A, \textsc{true}) \\
&\equiv \{\text{boolean algebra}\} \\
&\quad \neg\,\mathrm{wlp}(A, \textsc{false}) \wedge \mathrm{wp}(A, \textsc{true}) \\
&\Rightarrow \{A \sqsubseteq A'\}
\end{aligned}$$

$$\begin{aligned}
&\neg\, \mathrm{wlp}(A', \text{FALSE}) \wedge \mathrm{wp}(A', \text{TRUE})\\
\equiv\ &\{\text{pairing condition again; boolean algebra again}\}\\
&\neg\, \mathrm{wp}(A', \text{FALSE})\\
\equiv\ &\mathcal{G}(A')
\end{aligned}$$

The second argument to $\boxtimes$ is easier:

$$\begin{aligned}
&\mathrm{w(l)p}(A \boxtimes (\sqcup\, B :: B),\ R)\\
\equiv\ &\mathrm{w(l)p}(A, R) \wedge (\mathcal{G}(A) \vee \mathrm{w(l)p}((\sqcup\, B :: B),\ R))\\
\equiv\ &\mathrm{w(l)p}(A, R) \wedge (\mathcal{G}(A) \vee (\vee(\wedge)\ B :: \mathrm{w(l)p}(B, R)))\\
\equiv\ &\mathrm{w(l)p}(A, R) \wedge (\vee(\wedge)\ B :: (\mathcal{G}(A) \vee \mathrm{w(l)p}(B, R)))\\
\equiv\ &(\vee(\wedge)\ B :: \mathrm{w(l)p}(A, R) \wedge (\mathcal{G}(A) \vee \mathrm{w(l)p}(B, R)))\\
\equiv\ &(\vee(\wedge)\ B :: \mathrm{w(l)p}(A \boxtimes B,\ R))\\
\equiv\ &\mathrm{w(l)p}((\sqcup\, B :: A \boxtimes B),\ R)
\end{aligned}$$

The projection operator is easy:

$$\begin{aligned}
&\mathrm{w(l)p}([\![\, x \mid (\sqcup A :: A) \,]\!],\ R)\\
\equiv\ &(\forall x : \mathrm{w(l)p}((\sqcup A :: A),\ R))\\
\equiv\ &(\forall x : (\vee(\wedge)\ A :: \mathrm{w(l)p}(A, R)))\\
\equiv\ &(\vee(\wedge)\ A :: (\forall x : \mathrm{w(l)p}(A, R)))\\
\equiv\ &(\vee(\wedge)\ A :: \mathrm{w(l)p}([\![\, x \mid A \,]\!],\ R))\\
\equiv\ &\mathrm{w(l)p}((\sqcup A :: [\![\, x \mid A \,]\!]),\ R)
\end{aligned}$$

The assignment operator is continuous in all its command arguments, because it doesn't have any.

This completes the proof of the theorem. ∎

All of the pieces are now in place for the proof of the final theorem concerning the existence of fixpoints.

**Theorem 8.** *Let $f$ be a map from commands to commands defined by an expression of the form $f(X) \equiv \mathcal{E}$, where $\mathcal{E}$ is an expression built from the fundamental operators, assignment, projection, the command parameter $X$, and any number of fixed commands and predicates. Then $f$ has a least fixpoint under the $\sqsubseteq$ relation, equal to $(\sqcup i\ :\ i \geq 0\ :\ f^i(Loop))$, and to $(\sqcap X : f(X) \sqsubseteq X : X)$.*

*Proof.* Theorem 7 shows that the operators are continuous, therefore $f$ is continuous. Theorem 4 shows that any chain has a join in the approximation

order. Thus the conditions of the Limit Theorem are satisfied. The formula from the Limit Theorem justifies the first formula in the theorem, since, as is easily seen, the command *Loop* is $\sqsubseteq$-minimal. Thus $f$ has a fixpoint, so the meet in the theorem's second formula is non-empty, and therefore exists by Theorem 5, and equals $f$'s least fixpoint by the Knaster-Tarski Theorem. ∎

Note that the theorem equates the fixpoint both to an infinite meet and to an infinite join. Each equation has its uses, although in this paper we will use only the join equation.

No new difficulties are created by mutual recursion. For example, a mutually recursive definition of two commands $A$ and $B$ will have the form

$$(A, B) \equiv (f(A, B), g(A, B)),$$

where $f$ and $g$ are chain-continuous. This can be treated as a fixpoint equation over the universe of command pairs, for which a suitable approximation relation is defined by

$$(A, B) \sqsubseteq (A', B') \;\equiv\; A \sqsubseteq A' \;\wedge\; B \sqsubseteq B'.$$

The details are left to the interested reader.

Once mutual recursion is available, we can add **do-od** to the list of operators in the theorem, since iteration can be eliminated in favor of tail-recursion. The details of this elimination are the subject of the next section.

## 9 Iteration and tail-recursion

In this section, we will see that if **do-od** and **if-fi** are defined recursively, their semantics agree with Dijkstra's direct definitions. To show this, we will prove a general formula for the preconditions of a tail recursion.

We begin by comparing the two definitions of **do-od**. Since unrolling a loop doesn't change its meaning, the equation

$$\textbf{do}\ A\ \textbf{od} \;\equiv\; A\,;\textbf{do}\ A\ \textbf{od} \boxtimes \mathit{Skip}$$

is valid. Thus **do** $A$ **od** is a solution of the equation

$$X \;\equiv\; A\,; X \boxtimes \mathit{Skip}. \tag{1}$$

Appealing to Theorem 8, we define **do** $A$ **od** as the $\sqsubseteq$-least solution of the equation. Since the right-hand side is a chain-continuous function of $X$, the least solution exists.

Without Theorem 8, Hehner [10] proceeded as follows. For any $X$ satisfying (1), and any postcondition $R$, we have

$$\mathrm{w(l)p}(X,R) \;\equiv\; \mathrm{w(l)p}(A,\mathrm{w(l)p}(X,R)) \wedge (\mathcal{G}(A) \vee R) \tag{2}$$

by applying the precondition equations for the right-hand side. Thus w(l)p(**do** $A$ **od**, $R$) is a solution for $P$ of the predicate equation

$$P \;\equiv\; \mathrm{w(l)p}(A,P) \wedge (\mathcal{G}(A) \vee R)\,. \tag{3}$$

Hehner defined wp(**do** $A$ **od**, $R$) as the strongest solution of the equation, and wlp(**do** $A$ **od**, $R$) as the weakest solution of the equation. He also proved that this definition agreed with Dijkstra's original definition in *Discipline of Programming*, and Dijkstra has subsequently adopted Hehner's definition [5].

Note that Hehner's definition uses predicate recurrences; predicate transformer recurrences are not needed. His technique works because of the special form of (1): when the precondition of the right hand side of the equation with respect to $R$ is taken, the only occurrences of $X$ are in expressions of the form $\mathrm{w(l)p}(X,R)$. This allows the step from (2) to (3), where these occurrences are replaced by $P$. In other words, the technique works for the recursion $X \equiv f(X)$ provided that $\mathrm{w(l)p}(f(X),\ R)$ can be written as a function of $\mathrm{w(l)p}(X,R)$. We call such recursions *tail recursions*, and prove in the following theorem that Hehner's technique works for all of them, and therefore in particular for **do–od**.

**Theorem 9.** *Let $f$ be a chain-continuous totality-preserving function from commands to commands such that for some conjunctive predicate transformers $g$ and $gl$, the following identity holds for all predicates $R$ and all total commands $X$:*

$$\mathrm{w(l)p}(f(X),\ R) \equiv g(l)(\mathrm{w(l)p}(X,R),\ R)\,. \tag{4}$$

*Let $X_0$ be the $\sqsubseteq$-least fixpoint of $f$. Then for any predicate $R$,*

$$\mathrm{w(l)p}(X_0,R) \equiv \textit{the strongest (weakest) fixpoint of } g(l)(?,R)\,.$$

To apply the theorem to the semantics of **do** $A$ **od**, let

$$f(X) \equiv A\,;X \boxtimes \mathit{Skip}$$
$$g(l)(P,R) \equiv \mathrm{w(l)p}(A,P) \wedge (\mathcal{G}(A) \vee R)$$

Then the conditions of the theorem are satisfied. (Note that the assumption that $X$ is total is required to guarantee the equivalence of $\mathcal{G}(A\ ;\ X)$ and $\mathcal{G}(A)$.) The theorem implies that

$$\text{w(l)p}(\textbf{do}\ A\ \textbf{od},\ R) \equiv \text{the strongest (weakest) fixpoint of } g(l)(?,R),$$

which is exactly Hehner's formula for **do-od**.

The condition of Theorem 9 can be summarized by saying that $f$ is a chain-continuous total tail recursion. We will call (4) the tail recursion identity.

*Proof of Theorem 9.* To show that $\text{w(l)p}(X_0, R)$ is a fixpoint of $g(l)(?, R)$, observe that

$$\begin{aligned} & g(l)(\text{w(l)p}(X_0,R),\ R) \\ \equiv\ & \{\text{tail recursion identity}\} \\ & \text{w(l)p}(f(X_0),\ R) \\ \equiv\ & \{X_0 \text{ is a fixpoint of } f\} \\ & \text{w(l)p}(X_0,R) \end{aligned}$$

To show that $\text{w(l)p}(X_0, R)$ is the strongest (weakest) fixpoint, observe first that since $f$ is chain-continuous, Theorem 8 yields

$$X_0 \equiv (\sqcup\ n :: f^n(Loop))\ ,$$

and therefore, by Theorem 6,

$$\text{w(l)p}(X_0,R) \equiv (\vee(\wedge)\ n :: \text{w(l)p}(f^n(Loop),\ R))\ .$$

Let $P$ be any fixpoint of $g(?,R)$, then

$$\begin{aligned} & \text{wp}(X_0,R) \Rightarrow P \\ \equiv\ & (\vee\ n :: \text{wp}(f^n(Loop),\ R)) \Rightarrow P \\ \equiv\ & (\wedge\ n :: \text{wp}(f^n(Loop),\ R) \Rightarrow P) \\ \equiv\ & \{\text{induction below}\} \\ & \text{TRUE} \end{aligned}$$

The induction is easy: if $n = 0$, $\text{wp}(f^n(Loop),\ R)$ is FALSE, which is as strong as $P$. Also,

$$\begin{aligned} & \text{wp}(f^n(Loop),\ R) \Rightarrow P \\ \Rightarrow\ & \{g \text{ is monotonic}\} \end{aligned}$$

$$\begin{aligned}
& g(\mathrm{wp}(f^n(Loop),\ R),\ R) \Rightarrow g(P,R) \\
\equiv\ & \{\text{tail recursion}\} \\
& \mathrm{wp}(f^{n+1}(Loop),\ R) \Rightarrow g(P,R) \\
\equiv\ & \{P \text{ is a fixpoint of } g\} \\
& \mathrm{wp}(f^{n+1}(Loop),\ R) \Rightarrow P
\end{aligned}$$

The proof of the other half of the theorem is essentially the same, but the parenthesis convention is not powerful enough to allow the two halves to be folded together. Let $P$ be any fixpoint of $gl(?, R)$. Then

$$\begin{aligned}
& P \Rightarrow \mathrm{wlp}(X_0, R) \\
\equiv\ & P \Rightarrow (\wedge\ n :: \mathrm{wlp}(f^n(Loop),\ R)) \\
\equiv\ & (\wedge\ n :: P \Rightarrow \mathrm{wlp}(f^n(Loop),\ R)) \\
\equiv\ & \{\text{induction below}\} \\
& \text{TRUE}
\end{aligned}$$

The induction is easy: if $n = 0$, $\mathrm{wlp}(f^n(Loop),\ R)$ is TRUE, which is as weak as $P$. Also,

$$\begin{aligned}
& P \Rightarrow \mathrm{wlp}(f^n(Loop),\ R) \\
\Rightarrow\ & \{gl \text{ is monotonic}\} \\
& gl(P,R) \Rightarrow gl(\mathrm{wlp}(f^n(Loop),\ R),\ R) \\
\equiv\ & \{\text{tail recursion}\} \\
& gl(P,R) \Rightarrow \mathrm{wlp}(f^{n+1}(Loop),\ R) \\
\equiv\ & \{P \text{ is a fixpoint of } gl\} \\
& P \Rightarrow \mathrm{wlp}(f^{n+1}(Loop),\ R)
\end{aligned}$$

This completes the proof of the theorem. ∎

Tail recursion can be used to define **if-fi** as well as **do-od**. Let **if** $A$ **fi** be the $\sqsubseteq$-least solution of

$$X : X \equiv A \boxtimes X .$$

This equation can be justified by unrolling the operational definition "activate $A$ until it succeeds". Then Theorem 9 applies, and simple arguments yield the precondition equations for **if-fi**:

$$\begin{aligned}
& \mathrm{wp}(\textbf{if } A \textbf{ fi},\ R) \equiv \mathcal{G}(A) \wedge \mathrm{wp}(A, R) \\
& \mathrm{wlp}(\textbf{if } A \textbf{ fi},\ R) \equiv \mathrm{wlp}(A, R)
\end{aligned}$$

## 10 Concluding remarks

It is not difficult to show that the Limit Theorem remains true if the constraint on $f$ is weakened from "continuous" to "monotonic", and the range of $i$ in the expression $(\sqcup i :: f^i(\mathrm{min}))$ is changed from "the integers" to "the ordinals". Thus if any useful operations on commands turn up that are monotonic but discontinuous, they can still be included in recursive definitions, since Theorem 8 can be modified to rely on the more general form of the Limit Theorem.

A partially ordered set is *directed* if it contains an upper bound for every pair of its elements. A directed set is a generalization of a chain, and Theorems 4 and 6 are true for directed sets as well as chains.

The commands of the general calculus form a denotational semantic *domain*, that is, a continuous partial order. At least, this is true modulo some technicalities, which will now be explained. Let $A \ll B$ mean that for any chain, if $B$ approximates the join of the chain, then $A$ approximates some element of the chain. (For simplicity we continue to use chains instead of directed sets.) The formula $A \ll B$ is read "$A$ is way below $B$"; its operational meaning is that $A$ is one of the approximations that can be obtained by executing $B$ for a finite number of steps and classifying any uncompleted computations as loops. (This interpretation may be misleading if $B$ is unboundedly nondeterministic, but operational interpretations of unbounded nondeterminism are bound to be paradoxical.) The interpretation suggests that a command should equal the join of those commands that are way below it; if this is in fact true for all commands, the partial order $\sqsubseteq$ from which $\ll$ is constructed is said to be *continuous*.

A standard example of a discontinuous partial order is the implication order $\Rightarrow$ in a complete atomless predicate algebra. In such a predicate algebra, the approximation relation $\sqsubseteq$ constructed from $\Rightarrow$ will not be continuous, either. But I have proved that if $\Rightarrow$ is continuous (as it is in most predicate algebras of practical importance, including the algebra of boolean functions on a state space), then so is $\sqsubseteq$. In summary: if the underlying predicate algebra is continuous, the commands of the general calculus form a denotational semantic domain.

My final conclusion is that Dijkstra was right: general recursion is more complicated than simple iteration.

## Acknowledgements

I am grateful to Kathleen Sedehi, who typeset the paper; to Martín Abadi, Cynthia Hibbard, Jim Horning, and Lyle Ramshaw for helpful comments on the exposition; and to Mark Manasse and Lyle Ramshaw, for helpful discussions about the contents. Lyle suggested the fine word "outcome".

## References

[1] J.W. deBakker. Semantics and termination of nondeterministic recursive programs. *Automata, Languages, & Programming*, Edinburgh, 1976.
[2] E.W. Dijkstra. *A Discipline of Programming.* Prentice-Hall, 1976.
[3] Edsger W. Dijkstra. A personal summary of the Gries-Owicki theory (EWD554, 1976). Printed in Dijkstra's *Selected Writings on Computing: A Personal Perspective*, Springer-Verlag, 1982.
[4] E.W. Dijkstra and others. From predicate transformers to predicates. Manuscript EWD821. April, 1982.
[5] E.W. Dijkstra. Lecture notes "Predicate Transformers" (Draft). Manuscript EWD835. November, 1982.
[6] H. Egli. A mathematical model for nondeterministic computations. Zurich, ETH (1975). Cited by deBakker [1].
[7] Paul R. Halmos. *Algebraic Logic.* Chelsea, 1962.
[8] David Harel. *First-Order Dynamic Logic.* Lecture notes in computer science 68. Springer-Verlag 1979.
[9] David Harel and Vaughan R. Pratt. Nondeterminism in logic of programs (preliminary report). *POPL Proceedings*, Tucson, Arizona 1978.
[10] Eric C.R. Hehner. **do** considered **od**: a contribution to the programming calculus. *Acta Informatica* 11, 1979.
[11] Eric C.R. Hehner. Predicative Programming I. *CACM* 27 2, 1984.
[12] Leon Henkin, J. Donald Monk, and Alfred Tarski. *Cylindric Algebras*, North-Holland. Part I, 1971, Part II, 1985.
[13] C.A.R. Hoare. Programs are predicates. In *Mathematical Logic and Programming Languages*, edited by C.A.R. Hoare & J.C. Shepherdson, Prentice/Hall International, 1985.
[14] S.C. Kleene. *Introduction to Meta-mathematics.* D. Van Nostrand, Princeton, 1952.
[15] M.S. Manasse & C.G. Nelson. Correct compilation of control structures. Bell Labs Technical Memorandum, 1984.
[16] Zohar Manna. *Mathematical Theory of Computation.* McGraw-Hill, 1974.
[17] Greg Nelson. Predicate Algebra. Manuscript CGN5, 1983.
[18] David Lorge Parnas. A generalized control structure and its formal definition. *CACM* 26 8, April 1983.
[19] David L. Parnas. Technical Correspondence in *CACM* 28 5, 1985.
[20] A. Tarski. A lattice-theoretic fixpoint theorem and its applications. *Pacific J. Math* 5, 1955. Cited by [15].

## Index

alternative construct 8
approximation relation
  defined 1
  derived 34
  mentioned 4
assignment operation 22, 44
axiomatic semantics 7
axiomatic approach 3, 15

boolean properties, of predicates 14

Cartesian product 14
chain, defined 16
chain-continuity
  defined 16
  implies monotonicity 22
  of various operators 41
co-atomic predicate 19
cog 31
cograph correspondence 1, 31–33
com 31
commands-as-predicates approach 6
completeness property, of predicates 15
conjunctivity condition
  defined 18
  implies monotonicity 22
clairvoyant nondeterminism
  introduced 12
  example of 14
classical calculus 2
cylindric algebra 15
cylindrical properties, of predicates 15

deBakker 3
denotational semantics 7, 49
Dijkstra's calculus (also called classical calculus) 1, 2, 4–5
directed, defined for partially ordered set 49
domain 2, 49
**do–od** 8, 11, 13, 45, 46, 47
Dynamic Logic 4, 6

Egli, H. 3
Eindhoven Tuesday Afternoon Club 6, 28

*Fail* 10, 13, 19, 32
fixpoint method 1, 16, 33, 44
fundamental operators 11, 41

general calculus
  advantage of, over classical calculus 2
  approached via predicate transformers 17–22
  approached via relations 8–11
  approached operationally 11–14
general model (of Harel and Pratt) 5
guard, defined 8

Halmos 3, 15
Harel, David 3, 5
Havoc 33
Hehner, E.C.R. 1, 3, 6, 46, 47
HH model 6, 33
Hoare, C.A.R. 1, 3, 6
Hoare logic 5

if–fi operator 8, 10, 13, 45, 48
independence, defined in relation to
  commands 24, 28, 29
  predicates 15, 29
initializing guards 25

joins of chains 17, 35, 40

Knaster-Tarski Theorem 17, 38, 45

lambda-calculus 1, 7
Law of the Excluded Miracle
  defined 2
  mentioned 5, 6, 18
Limit Theorem 17, 45, 49
*Loop* 10, 13, 19, 32

monotonicity
  defined 16
  mentioned 11, 22, 49

nondeterminism 1, 2
  blind 11
  clairvoyant 11, 14
  unbounded 27, 49

operational approach (see also calculus of operations) 4
  to partial commands 11
Owicki-Gries theory 13

pairing condition, defined 18
parenthesis convention, defined 16
parsing
  as example of nondeterminism 14
partial commands
  defined 10
  introduced 2
  mentioned 4, 11, 13
partial correctness model 4–5
Parnas, David 5
polyadic algebra 15
Pratt, Vaughan 3, 5
predicate
  algebra 15
  boolean properties of 14
  cylindrical properties of 15
  defined 14
  independent of a variable 15
predicate transformer approach 4, 5, 14, 17, 19
predicative programming 6
projection operation 22, 44

relational approach (see also, calculus of relations) 4, 5, 8–11
repetitive construct 8

Scott, Dana 1, 3
*Skip* 8, 10, 11, 13, 19, 32
states, defined 14
subset relation 4, 11
substitution, concept defined 15

tail recursion 2, 45, 46, 47
Tarski 3, 15
total correctness 1
total correctness model 4, 6
  Parnas's variation on 5
  difference from general model 6
total relations, defined 4
type 15

variables
  defined 15
  type, of a 15

wlp 18
wp 18
wp-calculus 1

# SRC Reports

"A Kernel Language for Modules and Abstract Data Types."
R. Burstall and B. Lampson.
Research Report 1, September 1, 1984.

"Optimal Point Location in a Monotone Subdivision."
Herbert Edelsbrunner, Leo J. Guibas, and Jorge Stolfi.
Research Report 2, October 25, 1984.

"On Extending Modula-2 for Building Large, Integrated Systems."
Paul Rovner, Roy Levin, John Wick.
Research Report 3, January 11, 1985.

"Eliminating go to's while Preserving Program Structure."
Lyle Ramshaw.
Research Report 4, July 15, 1985.

"Larch in Five Easy Pieces."
J. V. Guttag, J. J. Horning, and J. M. Wing.
Research Report 5, July 24, 1985.

"A Caching File System for a Programmer's Workstation."
Michael D. Schroeder, David K. Gifford, and Roger M. Needham.
Research Report 6, October 19, 1985.

"A Fast Mutual Exclusion Algorithm."
Leslie Lamport.
Research Report 7, November 14, 1985.

"On Interprocess Communication."
Leslie Lamport.
Research Report 8, December 25, 1985.

"Topologically Sweeping an Arrangement."
Herbert Edelsbrunner and Leonidas J. Guibas.
Research Report 9, April 1, 1986.

"A Polymorphic $\lambda$-calculus with Type:Type."
Luca Cardelli.
Research Report 10, May 1, 1986.

"Control Predicates Are Better Than Dummy Variables For Reasoning About Program Control."
Leslie Lamport.
Research Report 11, May 5, 1986.

"Fractional Cascading."
Bernard Chazelle and Leonidas J. Guibas.
Research Report 12, June 23, 1986.

"Retiming Synchronous Circuitry."
Charles E. Leiserson and James B. Saxe.
Research Report 13, August 20, 1986.

"An $O(n^2)$ Shortest Path Algorithm for a Non-Rotating Convex Body."
John Hershberger and Leonidas J. Guibas.
Research Report 14, November 27, 1986.

"A Simple Approach to Specifying Concurrent Systems."
Leslie Lamport.
Research Report 15, December 25, 1986.

**Systems Research Center**
130 Lytton Avenue
Palo Alto, California 94301