General Correctness: a Unification of Partial and Total Correctness

Dean Jacobs[1]
David Gries[2]

TR 84-641
October 1984

Department of Computer Science
Cornell University
Ithaca, New York 14853

[1] This work was supported by the National Science Foundation under grant MCS-81-03605 and by the second author's Guggenheim Fellowship.
[2] Computer Science Department, University of Southern California. This paper is based on the first author's Ph.D. thesis at Cornell.
[3] Computer Science Department, Cornell University.

General Correctness: a Unification of Partial and Total Correctness[1]

Dean Jacobs[2] and David Gries[3]

[1] This work was supported by the National Science Foundation under grant MCS-81-03605 and by the second author's Guggenheim Fellowship.

[2] Computer Science Department, University of Southern California. This paper is based on the first author's Ph.D. thesis at Cornell.

[3] Computer Science Department, Cornell University.

Summary. General correctness, which subsumes partial and total correctness, is defined for both weakest preconditions and strongest postconditions. Healthiness properties for general-correctness predicate transformers are more uniform and complete than those for partial- and total-correctness systems. In fact, the healthiness properties for partial and total correctness are simple restrictions of those for general correctness. General correctness allows simple formulations of the connections between weakest and strongest postconditions and between the notions of weakest precondition under the "demonic" and "angelic" interpretations of nondeterminism. A problem that plagues *sp* —*sp*(P, C) is undefined if execution of C begun in some state of P may not terminate— disappears with the generalization.

This paper is a study of some simple theory underlying predicate transformer semantics, and as yet has little bearing on current programming practices. The theory uses a relational model of programs.

0. Introduction

Neither partial nor total correctness is adequate to distinguish among some obviously different programs. For example, consider the following three programs:

```
C0: skip
C1: do true → skip od
C2: if true → skip
    [] true → C1
    fi
```

For each program, the final state —if there is one— is equal to the initial state. *C0* always terminates, *C1* never terminates and *C2* may terminate but is not guaranteed to. The three programs are obviously different.

Partial correctness deals only with the relation between initial and final states of a program and cannot be used to express guaranteed termination;

using partial correctness, *C0* and *C2* are indistinguishable. Partial correctness can be expressed using ***weakest liberal preconditions wlp***: for program C and predicate P, $wlp(C, P)$ is a predicate that represents the set of all initial states such that execution of C, if it terminates, does so with P true. So $wlp(C0, P) = wlp(C2, P) = P$ for any predicate P.

Total correctness treats the same all programs that may not terminate; using total correctness, *C1* and *C2* are indistinguishable. Total correctness can be expressed using ***weakest preconditions wp***: for program C and predicate P, $wp(C, P)$ is a predicate that represents the set of all initial states such that execution of C is guaranteed to terminate with P true. So $wp(C1, P) = wp(C2, P) = false$ for any P. See [4] for details of *wp* and *wlp*.

Functions *wp* and *wlp* define sets of initial states of execution given desired sets of final states. One has similar functions *sp* and *slp*, which define sets of final states from sets of initial states. For predicate P and program C, the ***strongest liberal postcondition*** $wlp(P, C)$ represents the smallest set of final states such that execution of C, if it terminates, does so in one of them. The strongest postcondition $sp(P, C)$ represents the smallest set of final states such that execution of C begun in any state satisfying P is guaranteed to terminate in one of them.

The purpose of this paper is to generalize to functions $gwp(C, P)$ and $gsp(P, C)$ that do distinguish between programs such as *C0, C1* and *C2*. The theory is in terms of a ***relational model***, which is given in section 1. The generalization, in section 2, makes use of a fresh state $\perp$ that denotes the "state of nontermination". The extension not only allows the generalization but leads, quite naturally, to new functions *gwpa* and *gspa*. Function *gwpa* can be interpreted in terms of the ***angelic nondeterminism*** of Floyd [5], as opposed to the ***demonic nondeterminism*** used by Dijkstra in [4]. The relationships between the functions *gwp*, *gsp*, *gwpa*, *gspa* are given in Section 3.

When dealing with predicate transformers, one derives conditions, called healthiness properties, that must be satisfied for the system to be computationally meaningful. The healthiness properties for the general-correctness predicate transformers are given in section 4. They are more uniform than those for partial and total correctness. For example, *gsp*, unlike *sp*, is defined on all arguments. The partial- and total-correctness properties arise naturally by restricting those for general correctness (sections 5 and 6). Finally, in section 7 we consider restricted classes of programs, such as deterministic ones, and develop more properties of the general-correctness predicate transformers.

As yet, this work has had little bearing on programming practices. It is simply an attempt to provide a more uniform framework in which to view and relate the various notions of correctness. This seems to have been accomplished, and in a rather simple way. The use of a relational model has allowed to give very simple and almost mechanistic proofs. In fact, there is nothing deep or difficult in the paper.

As long as one is interested only in total correctness or in partial correctness, those systems should be used to reason about programs. However, at times one may need a system in which to reason more carefully about programs that may not terminate. One needs a general-correctness system. But this requires a predicate calculus that can express the ideas represented operationally here. Such a calculus was developed in the first author's thesis [12] and will be reported elsewhere. It may be helpful to say a few words about it here.

The calculus, in addition to allowing reasoning about the state $\perp$ of nontermination, must allow reasoning about undefined variables, since variables and expressions can be undefined in $\perp$. We choose a thre-valued logic, although this may not be the only approach. A small language is defined in terms of *gwp* using the calculus, and proofs of a few small programs are developed. The logic and the *gwp*-definitions are more complicated to use than a conventional system such as that of [4], because one has to worry about undefinedness —something we usually sweep under the rug. More experience and practice with such a system may help. Our work in this regard is similar in nature to that of Barringer, Cheng and Jones [3].

1. Predicates and the Relational Model of Programs

In our relational model, a program is represented by a set of pairs (s, s') for which execution of the program begun in state s can terminate in state s'. S denotes the set of possible initial and final states of a machine. In order to deal with nontermination, we add a fresh state $\perp$, so that the state space is the set $S_\perp = S \cup \{\perp\}$. The use of $\perp$ will become clear shortly.

Typically, one gives syntactic predicates (e.g. $b < c \wedge x = y$) to describe sets of states: a predicate represents the set of states in which it is true. With a relational model, however, one thinks of a predicate as *being* the set of states and dispenses with the syntactic formulation entirely. Thus, a predicate P is a subset of $S_\perp$. The fact that a predicate can contain $\perp$ is important for general correctness. Operations on predicates include $\cup$ (union), $\bigcup_i$ (union of sets indexed by i), $\cap$ (intersection), $\bigcap_i$ (intersection of sets indexed by i) and $\neg$ (complement with respect to $S_\perp$). We use two boolean-valued operators $\subseteq$ and $\underline{\cap}$, defined on subsets of $S_\perp$ as follows:

(1.0) **Definition.**
$$B \subseteq P \;\overset{\wedge}{=}\; \forall(s\colon s \in B\colon s \in P) \;=\; (B \cap P =)$$
$$B \,\underline{\cap}\, P \;\overset{\wedge}{=}\; \exists(s\colon s \in B\colon s \in P) \;=\; (B \cap P \neq \{\}).$$

Note that $\underline{\cap}$ is commutative and $\subseteq$ is not. Operator $\subseteq$ is conventionally read "is a subset of". Operator $\underline{\cap}$, read "has something in common with", is introduced so that certain algebraic properties can be easily recognized, in particular, the relationship between $\subseteq$ and $\underline{\cap}$:

(1.1) **Subset Theorem.** (a) $B \,\underline{\cap}\, P = \neg(B \subseteq \neg P)$
(b) $B \subseteq P = \neg(B \,\underline{\cap}\, \neg P)$.

Proof. (a) $B \sqcap P = \exists(s\colon s \in B\colon s \in P)$
$= \exists(s\colon s \in B\colon \neg(s \in \neg P))$
$= \neg\forall(s\colon s \in B\colon s \in \neg P)$
$= \neg(B \subseteq \neg P)$

(b) $B \subseteq P = \neg\neg(B \subseteq \neg\neg P)$
$= \neg(B \sqcap \neg P)$ (by (a) with $\neg P$ for P) □

A binary relation R on $S_\perp$ is a subset of $S_\perp \times S_\perp$. We use the following notations:

$s\ R\ s'$	$\triangleq (s, s') \in R$	
$s\ R$	$\triangleq \{s' \mid s\ R\ s'\}$	(*final image* of s wrt. R)
$R\ s'$	$\triangleq \{s \mid s\ R\ s'\}$	(*initial image* of s' wrt. R)
R^{-1}	$\triangleq \{(s', s) \mid s\ R\ s'\}$	(*inverse* of R)
$R1 \circ R2$	$\triangleq \{(s, s'') \mid \exists(s' :: s\ R1\ s' \ \wedge\ s'\ R2\ s''\}$	(composition of $R1$ and $R2$).

As mentioned earlier, a program is represented by a set of pairs (s, s'), i.e. by a binary relation on $S_\perp$. Several relational models have been proposed. Hoare and Lauer [11] represent a program by the set of all pairs of states (s, s') for which execution begun in s (has a nondeterministic choice that) leads to termination in s' —the possibility of non-termination is not recorded in the model. This is the simplest model for studying partial correctness, where nontermination is irrelevant, but it is inadequate for studying total correctness. Plotkin [14] proposes inserting the pair $(s, \perp)$ if execution begun in s may lead to non-termination. This model gives extraneous information for studying (only) total correctness: a pair (s, s') is irrelevant if there is also a pair $(s, \perp)$. Wand [17] proposes omitting pairs (s, s') for all s' if execution begun in s may lead to non-termination, and Smyth [16] proposes inserting pairs (s, s') for all s' if execution begun in s may lead to non-termination. The latter two approaches make the possibility of non-termination indistinguishable from the guarantee of non-termination. Smyth's approach is used implicitly by Hehner [9] in studying program specifications. Plotkin's approach is investigated by Guerreiro [7], de Bakker [2], de Roever [15] and many others.

We use Plotkin's model, since it is both necessary and sufficient for studying general correctness. A program C is represented by a binary relation R on $S_\perp$ as follows. Let the phrase "s may reach s'" mean that execution of C begun in s may (1) terminate in s' if $s' \neq \perp$ or (2) not terminate if $s' = \perp$. Then we have the

(1.2) **Definition.** *The binary relation R representing program C is*

$$R \triangleq \{(s, s') \mid s \text{ may reach } s'\}.$$

For example, the relations representing the three programs of section 0 are

C0: $\{(s, s) \mid s \in S_\perp\}$
C1: $\{(s, \perp) \mid s \in S_\perp\}$
C2: $\{(s, s) \mid s \in S_\perp\} \cup \{(s, \perp) \mid s \in S_\perp\}$.

Not all binary relations represent programs, since some do not exhibit features of computation. Following Guerreiro [7], we introduce *program relations* as relations that do exhibit (some of) the features:

(1.3) **Definition.** *A program relation R (on $S_\perp$) is a binary relation on $S_\perp$ that satisfies*

(a) $\forall(s\colon s \in S_\perp\colon s\,R \neq \{\})$
(b) $\perp R \subseteq \{\perp\}$
(c) $\forall(s\colon s\,R\ \mathit{infinite}\colon s\,R\,\perp)$.

Restriction (a) says that every initial state may reach some final state (which may be $\perp$) —execution must have some outcome and $S_\perp$ describes all possible outcomes. Restriction (b) says that execution "begun" in the state of non-termination does not terminate. Together, (a) and (b) mean that $\perp$ may reach $\perp$ and no other state. Restriction (c) allows unbounded nondeterminism only if there is the potential for non-termination. It is derived from the fact that in a finite amount of time we can guarantee choice between only a finite number of alternatives. Back [1] considers unboundedly nondeterministic constructs that always terminate.

These restrictions are necessary but not sufficient: some program relations do not represent any program. We use the restrictions in section 4 to derive the healthiness properties of general correctness, and we will argue that a binary relation is consistent with the healthiness properties iff it is a program relation.

Note that these restrictions refer to final images but not to initial images. We use this "asymmetry of computation" to explain in section 4 the differences between the healthiness properties of weakest preconditions and strongest postconditions.

2. Predicate Transformers

We develop definitions of the generalized weakest precondition (*gwp*) and generalized strongest postcondition (*gsp*), in much the same way that the definitions of *wp* and *sp* might be developed. We begin with the definition of a "Hoare triple" and derive *gwp* and *gsp* from it. The use of the operators $\subseteq$ and $\underline{\cap}$ leads us to two more predicate transformers, *gwpa* and *gspa*.

Consider predicates P, Q and relation R. The "Hoare triple" $\{P\}\,R\,\{Q\}$ is interpreted to mean "execution of the program represented by R begun in any state of P is guaranteed to reach some state of Q":

(2.0) **Definition.** *Let R be a relation and P and Q be predicates.*

$$\{P\}\,R\,\{Q\} \triangleq \forall(s\colon s \in P\colon s\,R \subseteq Q).$$

Note that $\{P\}\ R\ \{Q\}$ resembles a total-correctness triple if $\perp \in Q$ and a partial-correctness triple if $\perp \notin Q$. Definition (2.0) makes the negative restriction that P contain *only* initial states whose final image is a subset of Q. This requirement can also be expressed as the positive restriction that Q contain *all* final states whose initial image has something in common with P:

(2.1) **Alternative Formulation Theorem.**

$$\{P\}\ R\ \{Q\} \;=\; \forall(s' :: R\ s' \mathrel{\underline{\cap}} P \Rightarrow s' \in Q)$$

Proof. $\{P\}\ R\ \{Q\}$

$$\begin{aligned}
&= \forall(s :: s \in P \Rightarrow s\ R \subseteq Q)\\
&= \forall(s :: s \in P \Rightarrow \forall(s' :: s\ R\ s' \Rightarrow s' \in Q))\\
&= \forall(s,s' :: \neg(s \in P) \vee \neg(s\ R\ s') \vee s' \in Q)\\
&= \forall(s,s' :: (s\ R\ s' \Rightarrow s \in \neg P) \vee s' \in Q)\\
&= \forall(s' :: \forall(s :: s\ R\ s' \Rightarrow s \in \neg P) \vee s' \in Q)\\
&= \forall(s' :: R\ s' \subseteq \neg P \vee s' \in Q)\\
&= \forall(s' :: \neg(R\ s' \subseteq \neg P) \Rightarrow s' \in Q)\\
&= \forall(s' :: R\ s' \mathrel{\underline{\cap}} P \Rightarrow s' \in Q) \qquad \text{(by (1.1))} \ \square
\end{aligned}$$

We want to define predicate $gwp(R,\ Q)$ as the weakest (i.e. largest) P that satisfies $\{P\}\ R\ \{Q\}$. Using (2.0), we see that P may contain only states whose final image is a subset of Q, and $gwp(R,\ Q)$ should contain *all* these states. On the other hand, predicate $gsp(P,\ R)$ should be the strongest (i.e. smallest) Q that satisfies $\{P\}\ R\ \{Q\}$. Using (2.1), we see that Q must contain all states whose initial image has something in common with P, so $gsp(P,\ R)$ should contain *only* these states. So we have

(2.2) **Definition.** *Let R be a relation and P and Q be predicates.*

$$\begin{aligned}
gwp(R,\ Q) &\triangleq \{s \mid s\ R \subseteq Q\}\\
gsp(P,\ R) &\triangleq \{s' \mid R\ s' \mathrel{\underline{\cap}} P\}.
\end{aligned}$$

Weakest preconditions arise not so much from using $\subseteq$ as from using final images. Similarly, strongest postconditions arise not so much from using $\underline{\cap}$ as from using initial images. We define different notions of weakest precondition and strongest postcondition by interchanging these operators.

(2.3) **Definition.** *Let R be a relation and P and Q be predicates.*

$$\begin{aligned}
gwpa(R,Q) &\triangleq \{s \mid s\ R \mathrel{\underline{\cap}} Q\}\\
gspa(P,R) &\triangleq \{s' \mid R\ s' \subseteq P\}.
\end{aligned}$$

While gwp requires the final image of a state to be a subset of Q, $gwpa$ requires only that it have something in common with Q. gwp refers to Dijkstra's "demonic" interpretation of nondeterminism [4], in which the "worst" execution path may be chosen. Under this interpretation, one considers an initial state to be suitable (i.e. to be in $gwp(R,\ Q)$) iff execution is guaranteed to reach a state in Q. Hence, an implementation of demonic nondeterminism is free to choose any execution path.

On the other hand, $gwpa$ refers to Floyd's "angelic" interpretation of nondeterminism [5], in which the "best" execution path is chosen. An initial state is suitable (i.e. is in $gwpa(R, Q)$) iff at *at least one* execution path leads to a state in Q. An implementation of angelic nondeterminism uses backtracking or parallel evaluation.

Harel [8] presents a mathematical characterization of "execution method" in terms of trees of program states. Both interpretations of nondeterminism can be described in his framework; *gwp* and *gwpa* appear as the weakest precondition relativized to two different execution methods.

We might expect *gspa* to describe strongest postconditions under angelic nondeterminism. However, under angelic nondeterminism there is no unique strongest (smallest) postcondition for a given precondition, because two "candidates" for the strongest postcondition (in the sense that they cannot be made any stronger) can be completely disjoint.

Function *gspa* does not have an immediate relationship to a method of execution. While *gsp* requires the initial image of a state to have something in common with P, *gspa* requires it to be a subset of P. In other words, $gspa(P, R)$ is the set of all final states s' such that P contains *every* initial state that can reach s'.

3. The Relationships Between *gwp*, *gsp*, *gwpa*, and *gspa*

We show that there is a simple "inverse" relationship between *gwpa* and *gsp* and between *gspa* and *gwp*. We show that there is a simple "duality" relationship between *gwpa* and *gwp* and between *gspa* and *gsp*. The duality relationship describes the connection between weakest preconditions under the two interpretations of nondeterminism. The inverse and duality relationships together describe the connection between weakest preconditions and strongest postconditions. It is worth noting that the inverse/duality relationships are not as simple under partial or total correctness.

Remarkably, *gwpa*, the function that computes weakest preconditions under angelic nondeterminism, and *gsp*, the function that computes strongest postconditions under demonic nondeterminism, are almost identical. The only difference is that one maps sets of final states to sets of initial states and the other maps sets of initial states to sets of final states. Functions *gspa* and *gwp* have a similar relationship.

(3.0) **Inverse Theorem.** $gwpa(R, Q) = gsp(Q, R^{-1})$
$gspa(P, R) = gwp(R^{-1}, P)$.

Proof. $gwpa(R, Q) = \{s \mid s\, R \sqcap Q\} = \{s \mid R^{-1}\, s \sqcap Q\} = gsp(Q, R^{-1})$
$gspa(P, R) = \{s' \mid R\, s' \subseteq P\} = \{s' \mid s'\, R^{-1} \subseteq P\} = gwp(R^{-1}, P)$ $\square$

(3.1) **Corollary.** $gwp(R, Q) = gspa(Q, R^{-1})$
$gsp(P, R) = gwpa(R^{-1}, P)$.

(3.2) **Duality Theorem.** $gwpa(R, Q) = \neg gwp(R, \neg Q)$
$gspa(P, R) = \neg gsp(\neg P, R)$.

Proof. Simple manipulation, as in the proof of theorem (3.0). $\square$

(3.3) **Corollary.** $gwp(R,\ Q) = \neg gwpa(R, \neg Q)$
$gsp(P,\ R) \ = \neg gspa(\neg P, R)$.

The connection between weakest preconditions under the two interpretations of nondeterminism is given by (3.2) and (3.3). By (3.2), at least one execution begun in a state of $gwp(R,\ Q)$ reaches Q iff it is not the case that all execution paths reach $\neg Q$. By the corollary, all execution paths reach Q iff no execution path reaches $\neg Q$. Taken together, (3.0) and (3.2) allow us to formulate weakest preconditions in terms of stongest postconditions, and vice versa:

(3.4) **Reformulation Theorem.** $gwp(R,\ Q) \ = \neg gsp(\neg Q, R^{-1})$
$gwpa(R, Q) = \neg gspa(\neg Q, R^{-1})$.

(3.5) **Corollary.** $gsp(P,\ R) \ = \neg gwp(R^{-1},\ \neg P)$
$gspa(P, R) = \neg gwpa(R^{-1}, \neg P)$.

The results of this section are summarized in the following diagram.

$$\begin{tikzcd}
gwp(R,Q) = \{s \mid s\,R \subseteq Q\} \arrow[r, leftrightarrow, "Inverse"] \arrow[d, leftrightarrow, "Dual"'] & gspa(P,R) = \{s' \mid R\,s' \subseteq P\} \arrow[d, leftrightarrow, "Dual"] \\
gwpa(R,Q) = \{s \mid s\,R \underline{\cap} Q\} \arrow[r, leftrightarrow, "Inverse"] & gsp(P,R) = \{s' \mid R\,s' \underline{\cap} P\}
\end{tikzcd}$$

4. Healthiness Properties

Dijkstra [4] proposes certain healthiness properties that *wp* must satisfy if it is to be computationally meaningful. We begin this section by presenting corresponding properties of *gwp*. Included is a new property, called *strictness*, which controls the use of $\perp$. We then use theorem (3.4) to derive corresponding properties of *gsp*. The use of this theorem allows us to explain the differences between the properties of weakest preconditions and strongest postconditions. In particular, *gsp* satisfies a weaker version of certain properties because of the asymmetry of computation (which is exhibited by definition (1.3) of program relation).

Many discussions of axiomatic semantics do not present an operational model and therefore must *postulate* the healthiness properties. In our presentation, as in Hoare [10], these properties are *proven* in terms of our relational model. All properties apply to arbitrary binary relations on $S_\perp$, unless it is explicitly stated that they apply only to program relations.

(4.0) **gwp-Excluded Miracle.**

(a) $gwp(R,\ S_\perp) \ = S_\perp$
(b) $gwp(R,\ \{\}) \ = \{\}$ *for program relation R.*

Proof. (a) $gwp(R,\ S_\perp) = \{s \mid s\,R \subseteq S\} = S_\perp$

(b) $gwp(R,\ \{\}) = \{s \mid s\,R \subseteq \{\}\} = \{\}$ (by (1.0a)) $\square$

It is instructive to compare (4.0) to the equivalent formulations for total and partial correctness. Excluded Miracle for *wp* is written $wp(C, false) = false$, where predicate *false* represents the set $\{\}$. However, $wp(C, true) = true$ does not necessarily hold: $wp(C, true)$ represents the set of states that are guaranteed to lead to termination and depends on C. In a total-correctness system, *true* represents the set S and not $S_\perp$.

Excluded Miracle for *wlp*, on the other hand, is written $wlp(C,\ true) = true$, where *true* represents the set $S_\perp$. However, $wlp(C,\ false) = false$ does not necessarily hold: $wlp(C,\ false)$ represents the set of states that are guaranteed to lead to non-termination and depends on C. In a partial-correctness system, *false* represents the set $\{\perp\}$ and not $\{\}$.

Strictness (4.1), a new healthiness property, controls the use of $\perp$. It requires that execution "begun" in $\perp$ reach $\perp$, so that if $\perp \in gwp(R,\ Q)$ it had better be the case that $\perp \in Q$. Moreover, since *gwp* is a *weakest* precondition, if $\perp \notin gwp(R,\ Q)$ it had better be the case that $\perp \notin Q$.

(4.1) gwp-Strictness. $\perp \in Q = \perp \in gwp(R,\ Q)$ *for program relation R.*

Proof. $\perp \in Q = \perp R \subseteq Q$ (by (1.3))

$= \perp \in \{s \mid s\,R \subseteq Q\}$

$= \perp \in gwp(R,\ Q)$ $\square$

We sometimes refer to (4.1) in two parts:

(a) $\perp \in gwp(R,\ \{\perp\})$

(b) $\perp \notin Q \Rightarrow \perp \notin gwp(R,\ Q)$.

Properties (4.2) and (4.3) describe how *gwp* distributes across conjunction ($\cap$) and disjunction ($\cup$). They refer to an at-most-countably infinite set of predicates Q indexed by i.

(4.2) gwp-Conjunctivitis. $\bigcap_i gwp(R,\ Q_i) = gwp(R,\ \bigcap_i Q_i)$.

Proof. $\bigcap_i gwp(R,\ Q_i) = \bigcap_i \{s \mid s\,R \subseteq Q_i\}$

$= \{s \mid \forall(i{:}{:}\ s\,R \subseteq Q_i)\}$

$= \{s \mid s\,R \subseteq \bigcap_i Q_i\}$

$= gwp(R,\ \bigcap_i Q_i)$ $\square$

(4.3) gwp-Disjunctivitis. $\bigcup_i gwp(R,\ Q_i) \subseteq gwp(R,\ \bigcup_i Q_i)$.

Proof. Similar to that of (4.2). $\square$

(4.4) gwp-Monotonicity. If $Q1 \subseteq Q2$ then $gwp(R,\ Q1) \subseteq gwp(R,\ Q2)$.

Proof. Suppose $Q1 \subseteq Q2$. Then $Q1 \cap Q2 = Q1$, so that

$gwp(R,\ Q1) = gwp(R,\ Q1 \cap Q2)$

$= gwp(R,\ Q1) \cap gwp(R,\ Q2)$ (by (4.2))

$\subseteq gwp(R,\ Q2)$ $\square$

Property (4.4) was proven in terms of (4.2); (4.3) was proven from set theory but in fact follows from (4.4).

Define $gwp_R(Q) \overset{\Delta}{=} gwp(R, Q)$. We say that gwp_R is *continuous at* Q iff for every sequence of predicates $Q_0 \subseteq Q_1 \subseteq \ldots$ such that $\biguplus Q_i = Q$ we have $\biguplus gwp_R(Q_i) = gwp_R(\biguplus Q_i)$. This is pointwise continuity; we say that gwp_R is continuous if it is continuous at all Q. Theorem (4.5) shows that gwp_R is continuous at Q iff all ways of achieving Q (all t such that $t\,R \subseteq Q$) use bounded nondeterminism ($t\,R$ is finite). The intuition behind (one direction of) the proof is as follows. Suppose there is an initial state t that may reach an infinite number of final states, each in Q. Then there is a sequence of approximations to Q such that no approximation contains all the final states: a finite number of them are added at each stage. Hence, the final image of t is contained in Q but in none of the approximations to Q. Therefore, $t \in gwp_R(\biguplus Q_i)$, but $t \notin gwp_R(Q_i)$ and gwp_R is not continuous at Q. The following proof assumes that $S_\perp$ is at most countably infinite.

(4.5) **gwp-Pointwise Continuity.** *gwp_R is continuous at Q iff $\forall(t$: $t \in gwp_R(Q)$: $t\,R$ is finite$)$.*

Proof. Suppose $t\,R$ is infinite for some $t \in gwp_R(Q)$; note that $t\,R \subseteq Q$. Let $s_1, s_2, s_3, \ldots$ be an enumeration of $t\,R$. Define $Q_0 \overset{\Delta}{=} Q - t\,R$ and $Q_{i+1} \overset{\Delta}{=} Q_i \cup \{s_{i+1}\}$ for $0 \leq i$. Clearly, $Q_i \subseteq Q_{i+1}$ and $\biguplus Q_i = Q$. Now $\biguplus gwp_R(Q_i) = \biguplus \{s \mid s\,R \subseteq Q_i\} = \{s \mid \exists(i{::}\ s\,R \subseteq Q_i)\}$ doesn't contain t since no i satisfies $t\,R \subseteq Q_i$. But $t \in gwp_R(\biguplus Q_i) = gwp_R(Q)$, so $gwp_R(\biguplus Q_i) \neq \biguplus gwp_R(Q_i)$. Hence, gwp_R is not continuous at Q.

Now, suppose $t\,R$ is finite for all $t \in gwp_R(Q)$. Choose any sequence of predicates $Q_0 \subseteq Q_1 \subseteq \ldots$ such that $\biguplus Q_i = Q$. Now $t \in gwp_R(\biguplus Q_i)$ means $t\,R$ finite, so the following holds.

$$
\begin{aligned}
& t \in gwp_R(\textstyle\biguplus Q_i) \\
=\ & t \in \{s \mid s\,R \subseteq \textstyle\biguplus Q_i\} \\
=\ & t\,R \subseteq \textstyle\biguplus Q_i \\
=\ & \{\textit{since } t\,R \textit{ is finite and } Q_i \subseteq Q(i+1) \textit{ for all } 0 \leq i\} \\
& \exists(i{::}\ t\,R \subseteq Q_i) \\
=\ & t \in \{s \mid \exists(i{::}\ s\,R \subseteq Q_i)\} \\
=\ & t \in \textstyle\biguplus \{s \mid s\,R \subseteq Q_i\} \\
=\ & t \in \textstyle\biguplus gwp_R(Q_i)
\end{aligned}
$$

Therefore, $gwp_R(\biguplus Q_i) = \biguplus gwp_R(Q_i)$, so gwp is continuous at Q. □

By restriction (1.3c), programs can be unboundedly nondeterministic only if they have the potential for non-termination. Corollary (4.6) says that if all ways of achieving Q are guaranteed to terminate, so that none uses unbounded nondeterminism, then gwp_R is continuous at Q. Corollary (4.7) says that if non-termination is not permitted by Q, so that all ways of achieving Q are guaranteed to terminate, then gwp_R is continuous at Q.

(4.6) **Corollary.** *Let R be a program relation. If $\forall(t$: $t \in gwp_R(Q)$: $\neg(t\,R\,\perp))$, then gwp_R is continuous at Q.*

(4.7) **Corollary.** *If $\perp \notin Q$, gwp_R is continuous at Q for all program relations R.*

We have just given a set of healthiness properties that program relations satisfy. It is also true that any binary relation that is consistent with the healthiness properties is a program relation. Hence, definition (1.3) is necessary and sufficient to produce the healthiness properties. Hoare [10] points out that it is possible to give a healthy definition of a construct that is impossible (or impractical) to implement. Program relations that do not correspond to any program are associated with such definitions.

We now consider the healthiness properties of *gsp*. Corollary (3.5), $gsp(P, R) = \neg gwp(R^{-1}, \neg P)$, can be used to convert properties of *gwp* to corresponding properties of *gsp* —when both R and R^{-1} are program relations. This technique allows us to explain the differences between the properties of weakest preconditions and strongest postconditions. First, each property of *gwp* that applies to binary relations corresponds to a property of *gsp* that is its "reflection". This is understandable, given the $\neg$-operators in (3.5). Second, each property of *gwp* that applies only to program relations does not correspond to a property of *gsp*. This is understandable, given that (3.5) refers to R^{-1}, which may not be a program relation even if R is. We interpret this as follows. In order for *gsp* to satisfy such properties, R^{-1} has to satisfy the same restrictions as R, namely, restrictions (1.3). But this is not the case since these restrictions are asymmetric: they refer to final images and not initial images. The asymmetry of computation explains why *gsp* satisfies fewer properties than *gwp*.

All properties of *gsp* are stated without proof; see [12] for details. All properties apply to arbitrary binary relations on $S_\perp$ unless it is explicitly stated that they apply only to program relations.

(4.8) **gsp-Excluded Miracle.** $gsp(\{\}, R) = \{\}$.

(4.9) **gsp-Strictness.** $gsp(\{\perp\}, R) = \{\perp\}$ *for program relation R.*

(4.10) **gsp-Conjunctivity.** $\bigcap_i gsp(P_i, R) \supseteq gsp(\bigcap_i P_i, R)$.

(4.11) **gsp-Disjunctivity.** $\bigcup_i gsp(P_i, R) = gsp(\bigcup_i P_i, R)$.

(4.12) **gsp-Monotonicity.** *If $P1 \supseteq P2$, then $gsp(P1, R) \supseteq gsp(P2, R)$.*

Define $gsp_R(P) \triangleq gsp(P, R)$. We say that gsp_R is *continuous at P* iff for every sequence of predicates $P_0 \supseteq P_1 \supseteq \ldots$ such that $\bigcap_i P_i = P$ we have $\bigcap_i gsp_R(P_i) = gsp_R(\bigcap_i P_i)$. Function gsp_R is continuous if it is continuous at all P. Theorem (4.13) shows that gsp_R is continuous at P iff all final states that cannot be reached from a state in P can be reached by at most a finite number of other states.

(4.13) **gsp-Pointwise Continuity.** *gsp_R is continuous at P iff $\forall(t{:}\ t \notin gsp_R(P){:}\ R\,t$ is finite).*

5. Partial Correctness

We define partial correctness by restricting general correctness so that nontermination is always permitted: all predicates must contain $\perp$. This is done simply by restricting the domain of the predicate-argument of *gwp* and *gsp* to predicates containing $\perp$. Theorem (4.1) guarantees that if $\perp \in Q$ then $\perp \in gwp(R, Q)$, so that *gwp* has the appropriate range. Similarly, theorem (4.9) guarantees that if $\perp \in P$ then $\perp \in gsp(P, R)$, so that *gsp* has the appropriate range.

(5.0) **Definition.** *For program relation R and predicates P, Q containing* $\perp$,

$$wlp(R, Q) \triangleq gwp(R, Q)$$
$$slp(P, R) \triangleq gsp(P, R).$$

To see that (5.0) is reasonable, note that for any Q containing $\perp$ the conventional interpretation of *wlp* applied to the representation of $Q - \{\perp\}$ represents $gwp(R, Q) - \{\perp\}$. See Majster [13] for a relational characterization of *wlp*. Similarly, for any P containing $\perp$ the conventional interpretation of *slp* applied to the representation of $P - \{\perp\}$ represents $gsp(P, R) - \{\perp\}$.

Functions *wlp* and *gwp* enjoy the same healthiness properties, with two exceptions: (4.0b) and (4.1b) refer to predicates that do not contain $\perp$, so they cannot be formulated in the partial-correctness system.

(5.1) **wlp-Excluded Miracle.** $wlp(R, S_\perp) = S_\perp$.

(5.2) **wlp-Strictness.** $\perp \in wlp(R, \{\perp\})$ *for program relation R.*

(5.3) **wlp-Conjunctivitis.** $\bigcap_i wlp(R, Q_i) = wlp(R, \bigcap_i Q_i)$.

(5.4) **wlp-Disjunctivitis.** $\bigcup_i wlp(R, Q_i) \subseteq wlp(R, \bigcup_i Q_i)$.

(5.5) **wlp-Monotonicity.** *If* $Q1 \subseteq Q2$ *then* $wlp(R, Q1) \subseteq wlp(R, Q2)$.

If $\perp \in Q$ we say that wlp_R is *continuous at Q* iff for every sequence of predicates $Q_0 \subseteq Q_1 \subseteq \ldots$ such that $\perp \in Q_i$ for all $0 \leq i$ and $\bigcup_i Q_i = Q$ we have $\bigcup_i wlp_R(Q_i) = wlp_R(\bigcup_i Q_i)$. We can show wlp_R is continuous at Q ~~⇔~~ gwp_R is iff
continuous at Q, so

(5.6) **wlp-Pointwise Continuity.** wlp_R *is continuous at Q iff* $\forall(t: t \in wlp_R(Q): t\ R$ *is finite*$)$.

Functions *slp* and *gwp* enjoy the same healthiness properties, with one exception: (4.8) refers to predicates that do not contain $\perp$, so it cannot be formulated in the partial-correctness system. This means there is no formulation of Excluded Miracle for *slp*.

(5.7) **slp-Strictness.** $slp(\{\perp\}, R) = \{\perp\}$ *for program relation R.*

(5.8) **slp-Conjunctivitis.** $\bigcap_i slp(P_i, R) \supseteq slp(\bigcap_i P_i, R)$.

(5.9) **slp-Disjunctivitis.** $\bigcup_i slp(P_i, R) = slp(\bigcup_i P_i, R)$.

(5.10) **slp-Monotonicity.** *If $P1 \supseteq P2$ then $slp(P1, R) \supseteq slp(P2, R)$.*

If $\perp \in P$ we say that slp_R is *continuous at* P iff for every sequence of predicates $P_0 \supseteq P_1 \supseteq \ldots$ such that $\bigcap_i P_i = P$ we have $\bigcap_i slp_R(P_i) = slp_R(\bigcap_i P_i)$. Clearly, slp_R is continuous at P iff gsp_R is continuous at P, so

(5.11) **slp-Pointwise Continuity.** *slp_R is continuous at P iff $\forall(t$: $t \notin slp_R(P)$: $R\ t$ is finite$)$.*

The conventional formulation of (5.7) is $slp(false,\ C) = false$, which is *not* a formulation of Excluded Miracle.

6. Total Correctness

In analogy to the previous section, we define wp by restricting the second argument of gwp to predicates that do not contain $\perp$. Theorem (4.1) guarantees that the range of wp consists only of predicates that do not contain $\perp$. Function sp cannot be defined so simply, for gsp does not enjoy a strictness property like gwp. It is possible that $\perp \notin P$ but $\perp \in gsp(P, R)$. However, this happens precisely when sp is undefined, so we have

(6.0) **Definition.** *For program relation R and predicates P, Q not containing $\perp$,*

$$wp(R, Q) \triangleq gwp(R, Q)$$
$$sp(P, R) \triangleq \begin{cases} gsp(P, R) & \text{if } \perp \notin gsp(P, R) \\ undefined & otherwise. \end{cases}$$

Function gwp and wp enjoy the same healthiness properties, with two exceptions: (4.0a) and (4.1a) refer to predicates that contain $\perp$, so they cannot be formulated in a total-correctness system.

(6.1) **wp-Excluded Miracle.** $wp(R, \{\}) = \{\}$ *for program relation R.*

(6.2) **wp-Strictness.** $\perp \notin wp(R, Q)$ *for program relation R.*

(6.3) **wp-Conjunctivitis.** $\bigcap_i wp(R, Q_i) = wp(R, \bigcap_i Q_i)$.

(6.4) **wp-Disjunctivitis.** $\bigcup_i wp(R, Q_i) \subseteq wp(R, \bigcup_i Q_i)$.

(6.5) **wp-Monotonicity.** *If $Q1 \subseteq Q2$ then $wp(R, Q1) \subseteq wp(R, Q2)$.*

If $\perp \notin Q$ we say that wp_R is *continuous at* Q iff for every sequence of predicates $Q_0 \subseteq Q_1 \subseteq \ldots$ such that $\bigcup_i Q_i = Q$, we have $\bigcup_i wp_R(Q_i) = wp_R(\bigcup_i Q_i)$. Clearly, wp_R is continuous at Q iff gwp_R is continuous at Q. By Corollary (4.7), we have

(6.6) **wp-Pointwise Continuity.** *wp_R is continuous at all Q.*

Functions *sp* and *gsp* enjoy the same formulation of excluded miracle.

(6.7) **sp-Excluded Miracle.** $sp(\{\}, R) = \{\}$.

However, *sp* does not satisfy any of the other healthiness properties. For example, consider monotonicity. Suppose that for some R, every state in P leads only to termination and state s leads to non-termination. Note that $\perp \notin gsp(P, R)$, so $sp(P, R)$ is defined. But $\perp \in gsp(P \cup \{s\}, R)$, so $sp(P \cup \{s\}, R)$ is undefined. Since $P \cup \{s\} \supseteq P$ and it is not the case that $sp(P \cup \{s\}, R) \supseteq sp(P, R)$, monotonicity is violated.

7. Special Classes of Programs

We now investigate properties of *gwp* and *gsp* for the restricted classes of deterministic, terminating and invertible program relations.

(7.0) **Definition.** *Program relation R is* deterministic *if* $\forall(s :: |s\,R| \leq 1)$.

By (1.3a), R is deterministic iff $\forall(s :: |s\,R| = 1)$. Theorem (7.1) shows that the two interpretations of nondeterminism are equivalent for deterministic R.

(7.1) **Determinism Theorem.** *Program relation R is deterministic iff* $\forall(Q :: gwp(R, Q) = gwpa(R, Q))$.

Proof. $\forall(Q :: gwp(R, Q) = gwpa(R, Q))$
$$= \forall(s, Q :: s \in gwp(R, Q) = s \in gwpa(R, Q))$$
$$= \forall(s, Q :: s\,R \subseteq Q = s\,R \,\underline{\cap}\, Q)$$
$$= \forall(s :: |s\,R| \leq 1)$$
$= R$ is deterministic □

(7.2) **Corollary.** *Program relation R is deterministic iff* $\forall(Q :: gwp(R, Q) = gsp(Q, R^{-1}))$.

Proof. Immediate from (3.0). □

(7.3) **Corollary.** *Program relation R is deterministic iff* $\forall(Q :: gwp(R, \neg Q) = \neg gwp(R, Q))$.

Proof. Immediate from (3.2). □

Corollary (7.3) shows that *gwp* distributes across $\neg$ for deterministic R. Theorem (7.4) shows that *gwp* distributes across $\cup$ for deterministic R.

(7.4) **gwp-Deterministic Disjunctivitis.** *Program relation R is deterministic iff* $\biguplus_i gwp(R, Q_i) = gwp(R, \biguplus_i Q_i)$ *for all sequences* $Q_0, Q_1, \ldots$.

Proof. This can be proved using (4.11) and (7.2). □

The properties of *gsp* are unaffected by determinism; (7.0), like (1.3), restricts final images but not initial images. Let us now consider some restrictions of initial images.

A program relation is terminating if $\perp$ is the only state that may reach $\perp$:

(7.5) **Definition.** *Program relation R is* terminating *if* $R\bot \subseteq \{\bot\}$.

By (1.3), program relation R is terminating iff $R\bot = \{\bot\}$. The next theorem shows that *gsp* satisfies the stronger version of strictness for terminating R.

(7.6) **gsp-Terminating Strictness.** *Program relation R is terminating iff* $\forall(P{::}\ \bot \in P \ = \ \bot \in gsp(P,\ R))$.

Proof. $\forall(P{::}\ \bot \in P \ = \ \bot \in gsp(P,\ R))$

$= \forall(P{::}\ \bot \in P \ = \ R\bot \sqcap P)$ (by (2.2))

$= (R\bot = \{\bot\})$

$= R$ is terminating □

(7.7) **Corollary.** *R is terminating* $= \forall(P{:}\ \bot \notin P{:}\ sp(P, R)$ *is defined*$)$.

The corollary shows that *sp* can be used freely with terminating programs. Note that, by (1.3c), if R is terminating then it must be boundedly nondeterministic.

We can make stronger restrictions on initial images by considering programs that can be "run backwards".

(7.8) **Definition.** *Program relation R is* invertible *if R^{-1} is a program relation.*

Theorem (7.9) describes the conditions under which a program relation is invertible.

(7.9) **Invertibility Theorem.** *Program relation R is invertible iff*

(a) $\forall(s'{:}\ s' \in S_\bot{:}\ R\ s' \neq \{\})$

(b) $R\bot \subseteq \{\bot\}$

(c) $\forall(s'{:}\ s' \in S_\bot{:}\ R\ s'$ *is finite*$)$.

Proof. (7.9a) and (7.9b) are (1.3a) and (1.3b) applied to R^{-1}. (1.3c) applied to R^{-1} is $\forall(s'{:}\ R\ s'\ \textit{infinite}{:}\ \bot R\ s')$. By (1.3b), $\bot R\ s'$ iff $s' = \bot$, so $R\ s'$ must be finite for $s' \neq \bot$. By (7.9b) $R\ \bot$ is finite, so (7.9c) holds. □

Restriction (a) says that every final state may be reached from some initial state. Restriction (b) says that R is terminating. Restriction (c) says that R^{-1} must be boundedly nondeterministic.

Invertible program relations are symmetric with respect to the direction of computation. Therefore, *gwp* and *gsp* satisfy similar properties for invertible program relations. (7.6) shows that *gsp* satisfies the stronger version of strictness iff (7.9b) holds. Similarly, we can show that *gsp* satisfies the other law of excluded miracle, $gsp(S_\bot,\ R) = S_\bot$, iff (7.9a) holds. We remarked earlier that if R is terminating then it is boundedly nondeterministic. (7.9c) says that R^{-1} is also boundedly nondeterministic.

Suppose R is invertible, then $(s,\ s) \in R \circ R^{-1}$ for all $s \in S_\bot$. This is a characterization of one kind of invertibility: execution of the program composed with its inverse can make nondeterministic choices that reach the original starting state. We might describe this as the inverse under angelic nondeterminism. The conventional notion of invertibility can be characterized by further restricting R.

(7.10) **Definition.** *Invertible program relation R is* deterministically invertible *if* $\forall(s' :: |R\ s'| \le 1)$.

By (7.9a), invertible program relation R is deterministically invertible iff $\forall(s' :: |R\ s'| = 1)$. Hence, if R is deterministically invertible, then $s\ R \circ R^{-1} = \{s\}$ for all $s \in S_\perp$. This is a characterization of the conventional notion of invertibility: execution of the program composed with its inverse reaches the original initial state. Note that a deterministically invertible program relation may not be deterministic; (7.10) forbids two different initial states from reaching the same final state, but one initial state may reach several different final states.

(7.11) shows that the properties of *gsp* for deterministically invertible program relations are analogous to those of *gwp* for deterministic program relations. We state these properties without proof.

(7.11) **Deterministic Inversion Theorem.**

$$\begin{aligned} &R \text{ is deterministically invertible} \\ &\quad = \forall(P :: gsp(P, R) = gspa(P, R)) \\ &\quad = \forall(P :: gsp(P, R) = gwp(R^{-1}, P)) \\ &\quad = \forall(P :: gsp(\neg P, R) = \neg gsp(P, R)) \\ &\quad = (\textstyle\bigcap_i gsp(P_i, R) = gsp(\bigcap_i P_i, R) \text{ for all } P_0, P_1, \ldots). \end{aligned}$$

8. Summary

Neither partial nor total correctness is adequate to distinguish between some obviously different programs. We have introduced a notion of general correctness that subsumes partial and total correctness in an attempt to overcome this problem. And we have showed that the general-correctness properties are simpler, more uniform and more complete than the corresponding properties of partial and total correctness. Further, we have been able to exhibit more connections between weakest preconditions and strongest postconditions and between the two notions of weakest precondition under "demonic" and "angelic" interpretations of nondeterminism.

The "healthiness" properties developed here are precisely the rules one would need in trying to deal with general-correctness ideas in proving programs correct. However, no help has been given in this paper for dealing with that issue. We have been interested only in developing the "healthiness" properties —using a relational model— and not in showing how those properties can be put to use in developing programs. Some steps in this direction are in the first author's thesis [12], where a logic is developed that allows explicit reference to the undefinedness of variables and expressions. The reader is referred to [3], which also investigates a logic in which the undefined can be handled.

Acknowledgements. Thanks go to the Programming Research Group at Oxford University for their support and hospitality while this work was being done and to Jan Prins for many insightful and lengthy discussions. Thanks also to Fred Schneider for critical comments on an earlier draft.

References

[1] Back, R.J. Semantics of unbounded nondeterminism. *Proc. ICALP 80*, Lecture Notes in Computer Science 85, Springer Verlag, 1980.

[2] de Bakker, J.W. Recursive programs as predicate transformers. In *Formal Description of Programming Concepts* (E.J. Neuhold, ed.), Amsterdam, North Holland, 1978, pp. 165-181.

[3] Barringer, H., J.H. Cheng and C.B. Jones. A logic covering undefinedness in program proofs. Tech. Rep., University of Manchester, 1984.

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

[5] Floyd, R.W. Nondeterministic algorithms. *J. ACM 4* (1967), 636-644.

[6] Gries, D. *The Science of Programming*. Springer-Verlag, N.Y., 1981.

[7] Guerreiro, P. Another characterization of weakest preconditions. Lecture Notes in Computer Science 137, Springer-Verlag, N.Y., 1982, pp 164-177.

[8] Harel, D. On the total correctness of nondeterministic programs. IBM Research Report RC7691, 1979.

[9] Hehner, R. Predicative programming, Part I. *CACM 27* (1984), 134-143.

[10] Hoare, C.A.R. Some properties of predicate transformers. *J. ACM 23* (1978), 461-480.

[11] ___ and P.E. Lauer. Consistent and complementary formal theories of the semantics of programming languages. *Acta Informatica 3* (1974), 135-153.

[12] Jacobs, D. *General Correctness: a Unification of Partial and Total Correctness*. Ph.D. Thesis, Computer Science Dept., Cornell University, Fall 1984.

[13] Majster-Cederbaum, M.E. A simple relation between relational and predicate transformer semantics for nondeterministic programs. *Inf. Proc. Let. 4* (1980), 190-192.

[14] Plotkin, G.D. A powerdomain construction. *SIAM J. Computation 5* (1976), 452-487.

[15] deRoever, W.P. Dijkstra's predicate transformer, nondeterminism, recursion, and termination. Lecture Notes in Computer Science 45, Springer-Verlag, N.Y. (1976), pp. 472-481.

[16] Smyth, M. Powerdomains. *JCSS 16* (1978).

[17] Wand, M. A characterization of weakest preconditions. *JCSS 15* (1977), 209-212.