THE WEAKEST PRESPECIFICATION

by

C.A.R.Hoare and He, Jifeng

Technical Monograph PRG-44

June 1985

Oxford University Computing Laboratory
Programming Research Group
8-11 Keble Road
Oxford OX1 3QD
England

Oxford University Computing Laboratory
Programming Research Group
8-11 Keble Road
Oxford OX1 3QD
England

Contents

THE WEAKEST PRESPECIFICATION

C.A.R. Hoare and He, Jifeng

> "For, aside from the fact that the concepts occurring in this calculus possess an objective importance and are in these times almost indispensable in any scientific discussion, the calculus of relations has an intrinsic charm and beauty which makes it a source of intellectual delight to all who become acquainted with it. ALFRED TARSKI [6]

0. Introduction

In Dijkstra's calculus of weakest preconditions [2], a program is specified by a predicate R describing the desired properties of the values of the program's variables when the program successfully terminates. If Q is a program, the weakest precondition

$$wp\ (Q,R)$$

gives a predicate S which can be used as the most general specification of a program P which is to be executed before Q in order to guarantee that the combination P;Q will meet the original specification R. If

$$wp\ (Q,R) = true$$

then Q already meets the specification R, and the programming task is complete.

The weakest prespecification of program Q and specification R will be written

$$Q \setminus R$$

It serves the same purpose as Dijkstra's weakest precondition, but generalises it in four ways.

(1) The specification R describes not only the desired properties of the final values of the variables, but also their desired relationship with the initial values; such specifications are normal in specification languages like VDM[3] and Z[4].

(2) The parameter Q may be a program, or it may be just the specification of a program still to be written. Consequently, the weakest prespecification can assist in splitting a task R into two subtasks Q and Q\R; then any implementation of Q\R, when composed sequentially with any implementation of Q, is guaranteed to meet the original requirement R.

(3) A partial relation R is taken as the specification of a guarded command, whose guard must be contained in the domain of R. Thus guarded commands can be specified and implemented independently, before being collected into a set with if or do.

(4) The programming language is extended to include general recursion, of which iteration is a simple special case.

This increase in generality is obtained at the cost of some increase in complexity, which can be justified only when it is needed.

Section 1 of this paper proves the basic properties of the weakest prespecification within the framework of the relational calculus. Section 2 defines an extended version of Dijkstra's programming language; programs are a particular inductively defined subset of all relations, which are total; they enjoy a number of useful additional properties, including computability. Section 3 shows how weakest prespecifications and Dijkstra's programming language can be treated within the framework of VDM, in which a specification or a program is expressed in terms of a precondition as well as a relation. The more elaborate proofs are relegated to the appendix, where they are treated entirely within the framework of the relational

1. The calculus of relations

A relation is defined conventionally as a set of ordered pairs from some fixed universe Σ (usually left implicit)

$$R = \{s_o, s \mid s_o R s\}$$

The following notations are widely used

$\bar{R} = \{s_o, s \mid \neg(s_o R s)\}$	complement
$\breve{R} = \{s_o, s \mid s R s_o\}$	converse
$R;S = \{s_o, s \mid \exists \dot{s}.\ s_o R \dot{s} \wedge \dot{s} S s\}$	composition
$R \subseteq S = \forall s_o, s.\ s_o R s \longrightarrow s_o S s$	inclusion
$R \cup S = \{s_o, s \mid s_o R s \vee s_o S s\}$	or
$R \cap S = \{s_o, s \mid s_o R s \wedge s_o S s\}$	end
$\bigcup_n R_n = \{s_o, s \mid \exists n \in \mathbb{N}.\ s_o R_n s\}$	union
$\bigcap_n R_n = \{s_o, s \mid \forall n \in \mathbb{N}.\ s_o R_n s\}$	intersection
$O = \{s_o, s \mid \text{false}\}$	empty
$U = \{s_o, s \mid \text{true}\}$	universe
$I = \{s_o, s \mid s_o = s\}$	identity

We shall use the following elementary theorems of the theory of relations, without explicit reference:

$P;I = I;P = P$
$P;O = O;P = O$
$(P;Q);R = P;(Q;R)$
$U;U = U$

$(P \cup Q);R = (P;R) \cup (Q;R)$
$R;(P \cup Q) = (R;P) \cup (R;Q)$
$P \subseteq Q \wedge R \subseteq S \longrightarrow P;R \subseteq Q;S$
$P \subseteq P;U$

In this section we shall use relations to represent both programs and specifications. The inclusion reletion can therefore be interpreted in three differing ways. If P and Q are programs and R and S are specifications:

$P \subseteq S$ means that program P meets specification S, because everything it does satisfies S.

$P \leq Q$ means that program P is the same or more deterministic than Q; everything it does, Q can do too; but Q may do more. Further, P satisfies every specification satisfied by Q.

$S \subseteq T$ means that specification T is the same or weaker (more general) than S. Every program that satisfies S also satisfies T.

In Section 2, programs will be defined as a certain subset of total relations; but in this section, the distinction is irrelevant.

Suppose now we wish to develop a program to meet specification R, and we decide to achieve this by sequential composition of two program components. Then we decide that the specification of the second component will be Q. The next question is, what should be the specification of the first component? We would like to know the weakest (most general) specification whose satisfaction is both a necessary and sufficient condition for the correctness of the whole program. This will be called the weakest prespecification of Q to achieve R, and will be denoted by the infix backslash

$$Q \backslash R.$$

Readers who doubt the existence of this operator are requested to suspend their disbelief until the paragraph numbered (9) below.

Let P meet this specification, i.e.,

$$P \subseteq Q \backslash R$$

Then we want to ensure that P composed with any implementation of Q will meet the specification R. Because composition is monotonic, this is ansured if P composed with Q itself satisfies R, i.e.,

$$P;Q \subseteq R.$$

Thus we motivate the postulate

$$P \subseteq Q \backslash R \implies P;Q \subseteq R.$$

But we want $Q \backslash R$ to be the weakest relation for which this implication holds. Thus we need to strengthen the implication to an aquivalence

$$(P;Q) \subseteq R \equiv P \subseteq (Q \backslash R)$$

All the proofs in this section will be based on this single basic law for prespecifications.

In the arithmetic of natural numbers, we have a very similar law for the inverse of multiplication

$$n \times m \leq r \equiv n \leq r \div m$$

where division discards the remainder. In this analogy, all theorems proved from the basic law will hold true for natural number division. The identity relation corresponds to unity, the empty relation corresponds to zero, and the universal relation is a sort of infinity, as shown by the theorems

$$U = Q \backslash R = R \backslash U$$

Proof: from the basic law, since $U;Q \subseteq R$ and $U;R \subseteq U$.

The analogy with multiplication and division may be illuminating in what follows, where $n \cap m$ can be interpreted as the minimum of two numbers and $n \cup m$ as their maximum.

1.1 Algebraic properties

(1) $Q \backslash R$ itself is a solution to the original problem of finding an X such that

$$X;Q \subseteq R.$$

Proof: $Q \backslash R \subseteq Q \backslash R$

$$\therefore (Q \backslash R);Q \subseteq R$$

This result has the following consequences

(a) $U \backslash R \subseteq R$

Proof: $U \backslash R \subseteq (U \backslash R);U$

$\subseteq R$ by (1)

(b) $(U \backslash R);U = U \backslash R$

Proof: $((U \backslash R);U);U = (U \backslash R);(U;U)$

$= (U \backslash R);U$

$\subseteq R$ by (1)

$\therefore (U \backslash R);U \subseteq U \backslash R$ by the basic law

The reverse containment is trivial.

(2) P itself is a solution to the problem of finding an X such that

$$X;Q \subseteq P;Q$$

Proof: $P;Q \subseteq P;Q$

$$\therefore P \subseteq Q \backslash (P;Q)$$

(3) If $Q \backslash R$ contains the identity relation, then Q by itself meets the original specification R; this is a necessary and sufficient condition for completion of the programming task.

$$I \subseteq Q \backslash R \quad \equiv \quad Q \subseteq R$$

Proof: LHS $\equiv$ $1;Q \subseteq R$ $\equiv$ RHS

(4) Every program meets the specification that it behaves like itself

$$I \subseteq Q \backslash Q$$

(5) The identity relation gives no help in meeting any specification

$$P = I \backslash P$$

Proof: $X \subseteq P \equiv X;I \subseteq P$

$\equiv X \subseteq I \backslash P$

The introduction of X in this and in many later proofs enables the basic law to be used in the proof of equations as well as inequalities.

(6) In order to meet both specifications R and S with the aid of Q, it is necessary to write a program which both meets R with aid of Q, and also meets S with the aid of Q

$$Q \backslash (R \wedge S) = (Q \backslash R) \wedge (Q \backslash S)$$

Proof: $X \subseteq Q \backslash (R \wedge S) \equiv X;Q \subseteq R \wedge S$

$\equiv (X;Q \subseteq R) \wedge (X;Q \subseteq S)$

$\equiv (X \subseteq Q \backslash R) \wedge (X \subseteq Q \backslash S)$

$\equiv X \subseteq (Q \backslash R) \wedge (Q \backslash S)$

This law corresponds to one of Dijkstra's healthiness conditions for the weakest precondition

$$wp\ (Q,\ R \wedge S) = wp\ (Q,\ R) \wedge wp\ (Q,\ S)$$

The law extends to infinite conjunctions

$$Q \backslash (\bigcap_n R_n) = \bigcap_n (Q \backslash R_n)$$

(7) If P and Q are programs, $P \cup Q$ is a program which behaves either like P or like Q; and we cannot control the choice between them. In order to meet a specification reliably with the aid of $P \cup Q$, we must be prepared to meet it with the aid of P; we must also be prepared to meet it with the aid of Q

$$(P \cup Q) \backslash R = (P \backslash R) \wedge (Q \backslash R)$$

Proof: Similar to 6.

This law extends to infinite disjunctions:

(8) $(\bigcup_n P_n) \backslash R = \bigcap_n (P_n \backslash R)$

Proof: $X \subseteq (P \cup Q) \backslash R$

$\equiv X;(P \cup Q) \subseteq R$

$\equiv (X;P) \cup (X;Q) \subseteq R$

$\equiv (X;P) \subseteq R \wedge (X;Q) \subseteq R$

$\equiv X \subseteq R \backslash R \wedge X \subseteq Q \backslash R$

(8) In order to meet specification R with the aid of $(P;Q)$, you must ensure that P can meet R with the aid of Q

$$(P;Q)\backslash R \;=\; P\backslash(Q\backslash R)$$

Proof: $X \subseteq (P;Q)\backslash R \;\equiv\; X;P;Q \subseteq R$

$$\equiv\; X;P \subseteq Q\backslash R$$

$$\equiv\; X \subseteq P\backslash(Q\backslash R)$$

This law is the same as Dijkstra's definition of the weakest precondition for sequential composition

$$wp\ (P;Q,R) \;=\; wp\ (P, wp(Q,R))$$

(9) The time has surely arrived to check that the weakest prespecification actually exists. A simple definition is

$$Q\backslash R \;\triangleq\; \bigcup\{Y \mid Y;Q \subseteq R\}$$

and the basic law is readily proved from this definition

$$P;Q \subseteq R \;\longrightarrow\; P \in \{Y \mid Y;Q \subseteq R\}$$

$$\longrightarrow\; P \subseteq \bigcup\{Y \mid Y;Q \subseteq R\}$$

$$P \subseteq \bigcup\{Y \mid Y;Q \subseteq R\} \;\Longrightarrow\; P;Q \subseteq \bigcup\{Y;Q \mid Y;Q \subseteq R\} \quad ;\ \text{distrib } \bigcup$$

$$\longrightarrow\; P;Q \subseteq R \quad \text{set theory}$$

A weakest inverse can be similarly defined for any function which distributes through arbitrary unions. For conjunction, we have

$$P \wedge Q \subseteq R \;\equiv\; P \subseteq R \cup \overline{Q}$$

Disjunction does not have such an inverse, because it does not distribute through an empty union.

(10) A more explicit definition of the weakest prespecification can be given as a predicate relating an initial state s_0 with a final state $\dot{s}$

$$s_0(Q \backslash R)s \;=\; \forall \dot{s}.\; sQ\,\dot{s} \Longrightarrow s_0\, R\dot{s}$$

If $P = Q \backslash R$, this states that every possible final state s produced by P from initial state s_0, when used as input to Q, ensures that any final state $\dot{s}$ produced by Q is a final state allowed by R for the given initial state s_0. The above formula in the predicate calculus is useful when Q is a specification expressed as a predicate rather than a program expressed in a programming language.

The basic law may be proved by predicate calculus from this definition (or vice-versa)

$$\forall s.(s_0(P;Q)s \Longrightarrow s_0\, Rs)$$

$$\equiv \forall s.((\exists \dot{s}.\; s_0 P\dot{s} \wedge \dot{s}\, Qs) \Longrightarrow s_0\, Rs) \qquad \text{def ;}$$

$$\equiv \forall s.\forall \dot{s}.(s_0 P\dot{s} \wedge \dot{s}Qs \Longrightarrow s_0 Rs) \qquad \text{pred. logic}$$

$$\equiv \forall \dot{s}.(s_0 P\dot{s} \Longrightarrow \forall s(\dot{s}Qs \Longrightarrow s_0 Rs)) \qquad "$$

(11) The familiar concept of relational converse can be defined in terms of the weakest prespecification.

$$\breve{Q} \;=\; \bar{Q} \backslash \bar{I}$$

where $\bar{I}$ is the divereity relation (negation of identity).

i.e. $\bar{I} = \{ s_0, s \mid s_0 \neq s \}$

Proof:

$$s_0(\bar{Q} \backslash \bar{I})s \;\equiv\; \forall \dot{s}.(s\bar{Q}\dot{s} \longrightarrow s_0 \neq \dot{s}) \qquad \text{by (10)}$$

$$\equiv \forall \dot{s}(s_0 = \dot{s} \Longrightarrow s\bar{Q}\dot{s}) \qquad \text{predicate logic}$$

$$\equiv\; s\bar{Q}s_0 \qquad "$$

$$=\; s_0 \breve{Q} s \qquad \text{def } \breve{}$$

(12) Alternatively, the weakest prespecification can be defined in terms of the relational converse

$$Q \backslash R \;=\; \overline{\breve{R};\breve{Q}}$$

Proof:

$$\begin{aligned} s_0(\overline{\breve{R};\breve{Q}})s &\equiv \neg\,\exists \dot{s}.\; s_0 \bar{R} \dot{s} \wedge \dot{s} \breve{Q} s && \text{def ;} \\ &\equiv \forall \dot{s}\; \dot{s} \breve{Q} s \Rightarrow s_0 R \dot{s} && \text{pred. logic} \\ &\equiv s_0(\mathrm{LHS})s && \text{by (10)} \end{aligned}$$

(13) The next law states how negation distributes through weakest prespecification

$$\overline{Q \backslash R} \;=\; (\bar{R} \backslash \bar{Q}) \backslash \bar{I}$$

Proof:

$$\begin{aligned} s_0(\mathrm{RHS})s &\equiv \neg\, s(\bar{R} \backslash \bar{Q})s_0 && (11) \\ &\equiv \neg \forall \dot{s}.(s_0 \bar{R} \dot{s} \Rightarrow s \bar{Q} \dot{s}) && (10) \\ &\equiv \neg \forall \dot{s}(s Q \dot{s} \Rightarrow s_0 R \dot{s}) && \text{prop. logic} \\ &\equiv s_0(\mathrm{LHS})s && (10) \end{aligned}$$

This law might be called the second basic law for prespecifications, and it is needed in the proof of all laws quoted in the remainder of 1.1.

Since converse can be defined in terms of weakest prespecifications, it is an attractive idea to adopt the weakest prespecification as a primitive of the relational calculus in place of converse. In this case, the three Tarski axioms relating to converse can be replaced by the two basic laws for prespecifications. The details are worked out in the Appendix. Since P;Q is called the "relational product", its inverse could well be called the "relational quotient".

(14) In general, prespecification does not distribute through conjunction in its first argument. But if one of the limbs of the conjunction is of the form P;U, the following distribution law can be used

$$((P;U) \wedge Q) \setminus R \;=\; (P \setminus Q) \cup (Q \setminus R)$$

The relation P;U has the same domain as P, but relates each member of this domain to anything whatsoever

$$s_0 (P;U) s \;=\; \exists s' \cdot s_0 P s'$$

The relation $P \setminus Q$ is equal to $\overline{P;U}$; it relates everything to anything outside the domain of P

$$s_0 (P \setminus Q) s \;=\; \neg \exists s' \cdot s\, P\, s'$$

(15) A condition is written in a programming language as a boolean expression. We will represent such a condition as a relation whose domain is just those initial states in which the expression successfully evaluates to <u>true</u>, and which relates each member of its domain onto any final state whatsoever. More formally we define a relation b to be a condition if

$$b \;=\; b;U$$

We shall use lower case letters for conditions.

A guarded command in Dijkstra's programming notation is written $b \longrightarrow P$, where the guard b is a condition and P is a command. The guarded command is one that refuses to start unless its guard is true; we can therefore define it as a relation whose domain is restricted to the domain of b

$$b \longrightarrow P \;=\; b \wedge P$$

(16) If P is a relation with a restricted domain, it can be converted into a total relation by the operation

$$P^{+} = P \cup \overline{P;U}$$

If P represents a program, P^{+} behaves like P in all initial states in which P can start; but otherwise it does anything whatsoever. Its weakest prespecification is

$$P^{+}\backslash R = P\backslash R \cap (\overline{P\backslash U} \cup U\backslash R)$$

$U\backslash R$ is a condition describing those initial states in which the specification R permits anything whatsoever to happen

$$s_0(U\backslash R)s \equiv \forall \dot{s}.\ s_0 R \dot{s}$$

Proof: see appendix theorem 22.

(17) In Dijkstra's notation, guarded commands cannot occur individually but only in sets. For example, here is a set with two elements

$$\underline{\text{if}}\ b \longrightarrow P \,[]\, c \longrightarrow Q\ \underline{\text{fi}}$$

We can use the connective $\cup$ in place of the fat bar $[]$, and so represent the above command as

$$(b \cap P \cup c \cap Q)^{+}$$

(18) If F is a total function, we can prove

$$F\backslash R = R;(F\backslash I) \qquad \text{Proof: see appendix theorem 19.}$$

In the case of a total function

$$F\backslash I = \breve{F}$$

so the law quoted above gives an analogue of Dijkstra's weakest precondition for assignment

$$\begin{aligned} wp(a := F(a),\ s_0 R s) &= s_0 R(F(s)) && \text{Law of substitution} \\ &= \exists \dot{s}\ s_0 R \dot{s} \wedge a F \dot{s} && \text{predicate calculus} \\ &= s_0(R;\breve{F})s && \text{def } ; \end{aligned}$$

(19) The weakest postspecification of P with respect to R gives the weakest solution Q to the inequality

$$P;Q \subseteq R$$

when P and R are known in advance. It can be defined in terms of the weakest prespecification

$$R/P = (R \setminus \overline{I}) \setminus (P \setminus \overline{I})$$

and its properties are highly analogous, for example

$$R/(P;Q) = (R/P)/Q.$$

We leave the details as pleasant exercise for the interested reader.

1.2 Recursion and fixed points

A function F from relations to relations is said to be monotonic if it respects relational inclusion, i.e.,

$$X \subseteq Y \implies F(X) \subseteq F(Y) \qquad \text{for all } X \text{ and } Y$$

Tarski [5] has shown that the equation (over a complete lattice)

$$X = F(X)$$

has a solution whenever F is monotonic. In fact, there may be many solutions; the least of them will be denoted

$$\mu X.F(X).$$

It is defined by

$$\mu X.F(X) \triangleq \bigcap \{Y \mid F(Y) \subseteq Y\}$$

and is known as the least fixed point of the function F.

The importance of these results is that many useful operators of the relational calculus are monotonic in both their arguments, in particular the operators

$$\cup \quad ; \quad \cap$$

Furthermore, any expression made from monotonic operators is monotonic in all its operands. For example

$$F(X) = (b \cap (P;X)) \cup (\overline{b} \cap 1)$$

is monotonic in X. It corresponds to a program which first tests b; if b is true, it executes the command P, followed by X; if false, it terminates without changing anything. The construction

$$\mu X.F(X)$$

is the program which executes the whole of $\mu X.F(X)$ whenever called upon

In the above description to execute X. It therefore behaves like the conventional while-loop

$$\underline{\text{while}}\ b\ \underline{\text{do}}\ P$$

or, in Dijkstra's notation

$$\underline{\text{do}}\ b \longrightarrow P\ \underline{\text{od}}.$$

In the more general case $\mu X.F(X)$ is like a call on a recursive procedure, with name X and body $F(X)$; whenever X is encountered in the execution of the body, the whole of $\mu X.f(X)$ is executed in its place.

In addition to a least fixed point, a monotonic function also has a greatest fixed point, denoted $\nu X.F(X)$, which will be used in the next section.

The question now arises, what is the weakest prespecification of the least fixed point? The answer to this question requires more analysis.

A function F from relations to relations is said to be <u>continuous</u> if it distributes through the union of ascending chains of relations.

Definition. F is continuous if

$$F(\bigcup_{n\geq 0} R_n) = \bigcup_{n\geq 0} F(R_n)$$

whenever $\forall n \geq 0.\ \ R_n \subseteq R_{n+1}$

If F is continuous then it is monotonic, and its least fixed point can be constructed as the union of a series of approximations obtained by iterating the function F

$$\mu X.F(X) = \bigcup_{n\geq 0} F^n(0)$$

where $F^0(X) = X$

and $F^{n+1}(X) = F(F^n(X))$

From law 1(7)a, we can derive the weakest prespecification of $\mu X.F(X)$

$$(\mu X.F(X))\backslash R \;=\; \bigcap_{n\geq 0} (F^n(0)\backslash R).$$

The importance of this law derives from the fact that the operators

$$\cup \;;\quad \text{and} \quad \cap$$

are all continuous, and so is every function expressed by means of them. Consider for example the function F which defines iteration

$$F(X) \;=\; (b \cap (P;X)) \cup (\bar{b} \cap I)$$

For this function

$$\mu X.F(X)\backslash R \;=\; \bigcap_{n\geq 0} F_n$$

where $F_0 \;=\; U$

and $F_{n+1} \;=\; (b\backslash 0 \cup P\backslash F_n) \cap (\bar{b}\backslash 0 \cup R)$

Proof: By induction it is simple to prove that

$$F_n \;=\; F^n(0)\backslash R \qquad \text{for all } n.$$

Unfortunately, the law given above is not the same as Dijkstra's law for iteration, which would be

$$(\underline{\text{do}}\ b \longrightarrow P\ \underline{\text{od}})\backslash R \;=\; \bigcup_{n\geq 0} F_n$$

where $F_0 \;=\; \emptyset$

$$F_{n+1} \;=\; b \cap (P\backslash F_n) \cup \bar{b} \cap R$$

Something has gone seriously wrong in our attempt to use the relational calculus to explore Dijkstra's programming language. The nature of the problem and its cure will be explained in the next section.

2. Programming languages

In the previous section we developed a theory in which programs and specifications were represented by arbitrary relations. However, it is not possible to give a meaning to a recursively defined program unless the function defining the "body" of the recursive program is monotonic and preferably continuous. For example, negation is not monotonic, and there is no solution to the equation

$$X = \overline{X}$$

Consequently, the expressions

$$\mu X.\overline{X} \quad \text{and} \quad \nu X.\overline{X}$$

cannot have any reasonable meaning.

The easiest way to ensure that all recursively defined programs are meaningful is to restrict the notations of the programming language so that all the combinators of the language are continuous, or at least monotonic. This will also restrict the set of relations which are expressible in the language. That is the approach we shall take in this section.

There is another objective in restricting the notations of a programming language, namely the efficiency of implementation. The language $\mathcal{P}$ described in the next subsection does not achieve this objective, and is therefore perhaps more suitable for specification purposes than for practical implementation. The subsection 2.2 describes a language $\mathcal{D}$ which is a slight extension of the language of Dijkstra, and which can be implemented efficiently.

2.1 The language $\mathcal{P}$

The combinators and notations of the language $\mathcal{P}$ are the same as those of the relational calculus, but they exclude complement, converse, and weakest prespecification. More precisely, the language is the set of relations which can be defined from specified primitive relations by application of the permitted combinators.

Definition. $\mathcal{P}$ is the smallest set of relations satisfying the following conditions:

1. $0 \in \mathcal{P}$
2. If P is a total function, then $P \in \mathcal{P}$
3. If P and Q are in $\mathcal{P}$ then so are $P;Q$, $P \cup Q$, $P \cap Q$, and $b \cap P$, where b is a condition
4. If $P_i \in \mathcal{P}$ and $P_i \subseteq P_{i+1}$ for all i then $\bigcup_{i \geq 0} P_i$ is in $\mathcal{P}$

This language satisfies the major criterion of a programming language, that all the combinators are continuous, so that a recursively specified program always has a meaning

$$\mu X.F(X) = \bigcup_{n \geq 0} F^n(0)$$

and it satisfies the recursion equation

$$F(\mu X.F(X)) = \mu X.F(X),$$

However, there are serious problems in efficient implementation of the language $\mathcal{P}$.

The first problem is the combinator $\wedge$. This is an extremely powerful combinator in a programming language. For example, one could solve a sorting problem in the following simple fashion:

(1) Construct a program P which assigns to the array an arbitrary ascending sequence of numbers.

(2) Construct another program Q which applies to the array an arbitrary permutation.

(3) Then the program $P \wedge Q$ will sort the array into ascending sequence.

A program can be written like this in the programming language PROLOG, but it is extremely slow in execution, and a programmer would be recommended to transform such a program into one which does not use $\wedge$ in such a spectacular way.

The reason for the inefficiency is that an implementation must try all the possible executions of P and all possible executions of Q, and so find an execution of both of them which gives the same final result. The consequence is an exponential increase in the computing power required, at least in the worst case. Furthermore, if $\wedge$ is used elsewhere in the program, it will often be necessary to discover <u>all</u> matching executions of P and Q. The implementation does not know how many there are, and may have to go on looking forever. After a few recursions, an exponentially growing proportion of the available computing power is expended on these fruitless searches. In PROLOG, the cut is frequently used to keep this problem under control.

Another feature of the language is that there is no obligation on the programmer to write recursions that terminate. A non-terminating

recursion (e.g., $\mu X.X$) is equal to the empty relation 0. It follows that for any program P

$$(\mu X.X) \cup P = P$$

But an implementation cannot detect *which* of the operands of $\cup$ is going to fail to terminate; so both operands must be executed simultaneously (or in turns) until one of them terminates. If all results of execution are required, computing power will continue to be wasted on the nonterminating calculation. Even PROLOG shies away from this inefficiency by giving precedence to the first alternative, thereby departing from the ideal of a strictly logical semantics.

In conclusion, the language $\mathcal{P}$ is highly unsuitable for efficient implementation. It might be useful as a language for *specification* and design; but for these purposes recursion does not play such a central role, and there is less reason to accept an embargo on such highly expressive concepts as negation. In spite of its elegance, simplicity and power, the language $\mathcal{P}$ is not much use, and we shall pursue it no further.

2.2 The language $\mathcal{P}$

In this section we define a language $\mathcal{D}$ which avoids the problems of $\mathcal{P}$ at the expense of slightly greater complication. This language simply omits the problematic combinator $\cap$. The problem of $\cup$ is solved by ensuring that all relations P of the language are total, i.e.,

$$P;U = U.$$

An indirect consequence of this is that the program

$$P \cup Q$$

never requires execution of more than one of P or Q, and the implementation may choose either of them arbitrarily. This kind of arbitrary choice is sometimes known as "demonic non-determinism", because the programmer must be prepared for an implementation which is both omniscient and malevolent, and so will choose the least favourable alternative whenever the program allows such a choice. The nondeterminism of the language $\mathcal{P}$ is "angelic" in the sense that it will give a result whenever a result is possible, even at the expense of an exponential increase in consumption of computing resources.

In the language $\mathcal{P}$, a partial relation R represents a program which, if started in a state outside its domain, would fail to terminate. Since all relations in $\mathcal{D}$ are total, it is not possible to represent non-termination in this way. Instead we introduce a fictitious state $\perp$ into the domain and range of all relations in $\mathcal{D}$. This stands for the (forever unreachable) "result" of a program that never terminates. If P is a program, the inverse image of $\perp$ is the set of initial states for which P may fail to terminate; so its complement is the set of initial states in which P must terminate. If any program is started in this fictitious state (which of course it never will be), then we decree arbitrarily that the final state will be wholly arbitrary.

The introduction of $\perp$ to totalise a partial function is a common mathematical construction. For example, if you want to calculate how many partial functions there are between finite sets A and B, the easiest way is to extend B by a fictitious element $\perp$, and then compute the number of <u>total</u> functions between A and $B \cup \{\perp\}$. In other branches of mathematics, all the basic concepts are just as fictitious as $\perp$ - you can never observe points, lines, singularities, infinities, etc. In our theory $\perp$ plays a particularly important role; its introduction gives a method of specifying and proving that a program <u>cannot fail</u> to terminate. To prove the absence of error, it is necessary to use a mathematical theory in which the error is explicitly represented.*

Because all relations of the language $\mathcal{D}$ are total, the empty relation 0 is excluded. It is necessary to choose another representation for a nonterminating recursion. An appropriate choice is the universal relation U. One of the troubles with 0 in the language $\mathcal{P}$ is that it satisfies <u>every</u> specification R

$$0 \subseteq R \qquad \text{for all } R$$

and so acts as a "miracle" in the sense of Dijkstra. Yet the non-terminating program is in practice among the least satisfactory programs - it would hardly even satisfy a customer who doesn't care what he gets! That is why we identify the non-terminating program with U, i.e., a program that can do anything whatsoever - even fail to terminate. U may seem to be worse than a non-terminating program, because it may actually terminate with a wrong result. But in Dijkstra's view it is the programmer's responsibility to avoid non-termination as well as every other error, so there is no need to make distinctions between the various ways in which a program may go wrong.

---

* Readers with good taste may still find it difficult to accept $\perp$. They should be comforted by the fact that our whole theory can be developed without $\perp$, at the expense of complicated new definitions of all the relational operators, as shown in section 3.

There is another problem: the least fixed point of a function on total relations is not necessarily total. The solution is to use the greatest fixed point instead of the least, and replace the requirement for continuity of the combinators of the language by its dual, co-continuity. A function F is said to be co-continuous if it distributes through the intersection of descending chains of programs.

Definition. F is co-continuous if

$$F(\bigcap_{i\geq 0} P_i) = \bigcap_{i\geq 0} F(P_i)$$

whenever $P_i \in \mathcal{D}$ and $P_{i+1} \subseteq P_i$ for all i. Provided this condition is met, the greatest fixed point can be defined as an intersection

$$\nu X.F(X) = \bigcap_{n\geq 0} F^n(U)$$

and this satisfies the recursion equation

$$F(\nu X.F(X)) = \nu X.F(X).$$

With these explanations, the time has come to define the language $\mathcal{D}$. It is the smallest set of relations satisfying the conditions

1. $U \in \mathcal{D}$,
2. If P is a total function on every state except $\perp$ then $P^+ \in \mathcal{D}$
3. If P and Q are in $\mathcal{D}$, then so are $P;Q$ and $(b \cap P \cup c \cap Q)^+$ (where b and c are conditions)
4. If $P_i \in \mathcal{D}$ and $P_{i+1} \subseteq P_i$ for all $i \geq 0$ then $(\bigcap_{i\geq 0} P_i) \in \mathcal{D}$.

Dijkstra's language is contained in $\mathcal{D}$, as shown by the following correspondences:

$$\underline{\text{abort}} \;=\; U$$

$$\underline{\text{skip}} \;=\; \{\langle s_0, s\rangle \mid s = s_0 \wedge s_0 \neq \perp\}^+$$

$$(s := f(s)) = \{\langle s_0, s\rangle \mid s = f(s_0) \wedge s_0 \neq \perp \wedge s \neq \perp\}^+$$

$$P;Q \;=\; P;Q$$

$$\underline{\text{if}}\ b \longrightarrow P \,[]\, c \longrightarrow Q\ \underline{\text{fi}} \;=\; (b \wedge P \cup c \wedge Q)^+$$

$$\underline{\text{do}}\ b \longrightarrow P\ \underline{\text{od}} \;=\; \mu X.\ \underline{\text{if}}\ b \longrightarrow (P;X) \,[]\, \neg b \longrightarrow \underline{\text{skip}}\ \underline{\text{fi}}$$

where $\neg b$ has the same domain as b, and each is true in those states in which the other is false.

An assignment to a subset of the variables of a state can readily be extended to an assignment over the whole state. The definition of guarded command sets can readily be adapted to any finite number of alternatives.

The claim that this language is similar to Dijkstra's is supported by an operational understanding of that language; it will be strengthened in the next subsection by the similarity between the weakest prespecification of each language construct and Dijkstra's weakest precondition. Meanwhile it remains to prove the claim that all relations in $\mathcal{D}$ are total. To do this, it is necessary to choose a slightly stronger induction hypothesis, which includes a "finitary" property. This property is also needed in proving the second claim, that all the combinators of the language are co-continuous. These proofs will be completed in the remainder of this subsection.

Let the set of all states (including $\perp$) be Σ.

For any subset B of Σ and any relation P, we define $B \upharpoonright P$ as the image under P of these initial states in B.

$$B \upharpoonright P = \{s \mid \exists s_0 \in B \;.\; s_0 \; P \; s\}$$

P is defined as a total finitary relation if for any $s \in \Sigma$, $\{s\} \upharpoonright P$ is either a nonempty finite set or the universal set, Σ; and furthermore $\{\perp\} \upharpoonright P$ is universal. For future use we define $k = \perp \times \Sigma$

First the following results are trivial:

$$\Sigma \upharpoonright P = \Sigma \qquad \text{if P is finitary}$$
$$B \upharpoonright (P \cup Q) = (B \upharpoonright P) \cup (B \upharpoonright Q)$$
$$B \upharpoonright (P;Q) = \overline{(B \upharpoonright P) \upharpoonright Q}$$
$$\{s\} \upharpoonright P = (\{s\} \upharpoonright \breve{P})$$
$$\{s\} \upharpoonright (\bigcap_i P_i) = \bigcap_i (\{s\} \upharpoonright P_i)$$
$$B \upharpoonright (b \wedge P) = B_b \upharpoonright P \qquad \text{where } B_b = \{s \mid s \in B \wedge b \text{ is true in } s\}$$

Now we intend to show that all programs in $\mathcal{D}$ are total finitary relations.

(1) U is total finitary

Proof: $s \in \Sigma \longrightarrow \{s\} \upharpoonright U = \Sigma$

(2) If P is a total function, than P ∪ k is total finitary.

Proof: $s \neq \perp \longrightarrow \{s\} \upharpoonright (P \cup k) = \{s\} \upharpoonright P \cup \{s\} \upharpoonright k$
$= \{s\} \upharpoonright P$

which has exactly one member, because P is a total function.

Since $\perp$ is not in the domain of P, it follows that

$$\{\perp\}\restriction(P \cup k) = \{\perp\}\restriction P \cup \{\perp\}\restriction k = \{\perp\}\restriction k = \Sigma$$

(3) Let P and Q be total finitary relations. Then P;Q is also total finitary.

Proof: case 1 $\{s\}\restriction P = \Sigma$

$$\{s\}\restriction(P;Q) = (\{s\}\restriction P)\restriction Q = \Sigma\restriction Q = \Sigma$$

case 2 $\{s\}\restriction P = \{s_1, s_2, \ldots, s_m\}$

$$\{s\}\restriction(P;Q) = (\{s\}\restriction P)\restriction Q = \{s_1, s_2, \ldots, s_m\}\restriction Q$$
$$= \bigcup_I (\{s_i\}\restriction Q)$$

either $\exists i.(\{s_i\}\restriction Q = \Sigma)$ and then $\bigcup_i \{s_i\}\restriction Q = \Sigma$

or $\forall i.(\{s_i\}\restriction Q$ is finite), and so is $\bigcup_i \{s_i\}\restriction Q$.

(4) If P and Q both are total finitary, and b and c are conditions, then $(b \wedge P \cup c \wedge Q)^+$ is also finitary.

Proof:

$$\{s\}\restriction(b \wedge P \cup c \wedge Q)^+ = \{s\}\restriction(b \wedge P) \cup \{s\}\restriction(c \wedge Q) \cup \{s\}\restriction(\bar{b} \wedge \bar{c})$$ theorem 21 in appendix

Since P and Q are total, at least one of these terms is non-empty. Since each of them is either finite or equal to Σ, so is their union.

(5) If for all i, P_i is total finitary and $P_{i+1} \subseteq P_i$, then $(\bigcap_i P_i)$ is total finitary.

Proof: For any $s \in \Sigma$ we have

$$\{s\}\restriction(\bigcap_i P_i) = \bigcap_i(\{s\}\restriction P_i)$$

Thus it follows that

$$\forall i.\ (\{s\}\restriction P_i = \Sigma) \Longrightarrow (\{s\}\restriction (\bigcap_i P_i) = \Sigma)$$

Whenever there exists i_o such that $\{s\}\restriction P_{i_o} = \{s_1, \ldots, s_m\}$, if for all $n > i_o$ $\{s\}\restriction P_n = \{s\}\restriction P_{i_o}$, then we know that

$$\{s\}\restriction (\bigcap_i P_i) = \{s_1, \ldots, s_m\}.$$

Otherwise there exists $i_1 > i_o$ such that $\{s\}\restriction P_{i_1} \subset \{s\}\restriction P_{i_o}$, and then we repeat the same argument for i_1 as above for i_o. Since $\{s_1, \ldots s_m\}$ is finite, the above process will be finite before we find a i_N such that for all $n > i_N$

$$\{s\}\restriction P_n = \{s\}\restriction P_{i_N}$$

This completes the proof.

By the results (1) - (5) just proved, we have shown that all programs in $\mathfrak{D}$ are total finitery. Now we turn to examine the continuity of these operators in $\mathfrak{D}$.

(6) If Q is a total finitary relation, and $\{P_i\}$ is a decreasing chain of total finitary relations, b and c are conditions, then

$$(b \wedge (\bigcap_i P_i) \cup c \wedge Q)^+ = \bigcap_i (b \wedge P_i \cup c \wedge Q)^+$$

Proof: By (5) it follows that $(\bigcap_i P_i)$ is also total finitery. Furthermore we have

$$\text{LHS} = b \wedge (\bigcap_i P_i) \cup c \wedge Q \cup \bar{b} \cap \bar{c} \qquad \text{theorem 21}$$

$$= (\bigcap_i (b \wedge P_i)) \cup (c \wedge Q) \cup (\bar{b} \cap \bar{c}) \qquad \text{(set theory)}$$

$$= \bigcap_i (b \wedge P_i \cup c \wedge Q \cup \bar{b} \wedge \bar{c}) \qquad \text{(set theory)}$$

$$= \text{RHS} \qquad \text{theorem 21}$$

(7) If $\{P_i\}$ is a decreasing chain of total finitary relations, then

$$(\bigcap_i P_i);Q = \bigcap_i (P_i;Q)$$

Proof:

$$\forall i\ (\{s\}\uparrow P_i = \Sigma) \Longrightarrow \{s\}\uparrow((\bigcap_i P_i);Q) = (\{s\}\uparrow(\bigcap_i P_i))\uparrow Q$$

$$= (\bigcap_i (\{s\}\uparrow P_i))\uparrow Q = (\bigcap_i \Sigma)\uparrow Q$$

$$= \Sigma\uparrow Q = \bigcap_i (\Sigma\uparrow Q) = \bigcap_i ((\{s\}\uparrow P_i)\uparrow Q)$$

$$= \{s\}\uparrow \bigcap_i (P_i;Q)$$

$$\exists i\ (\{s\}\uparrow P_i \text{ is finite}) \Longrightarrow \exists i_o.\forall n.\ ((n \geqslant i_o) \Longrightarrow (\{s\}\uparrow P_n = \{s\}\uparrow P_{i_o})) \quad (5)$$

$$\Longrightarrow \{s\}\uparrow((\bigcap_i P_i);Q) = (\{s\}\uparrow(\bigcap_i P_i))\uparrow Q$$

$$= (\bigcap_i (\{s\}\uparrow P_i))\uparrow Q = (\{s\}\uparrow P_{i_o})\uparrow Q$$

$$= \{s\}\uparrow(P_i;Q) \quad \text{for } i \geqslant i_o$$

$$= \bigcap_{i \geqslant i_o} (\{s\}\uparrow(P_i;Q)) = \{s\}\uparrow \bigcap_i (P_i;Q)$$

(8) If P is a total finitary relation, and $\{Q_i\}$ is a decreasing chain of total finitary relations, then

$$P;(\bigcap_i Q_i) = \bigcap_i (P;Q_i)$$

Proof: Case 1. $\{s\}\backslash P = \Sigma$

$$\{s\}\backslash(P;\bigcap_i Q_i) = \Sigma\backslash\bigcap_i Q_i = \Sigma$$

$$= \bigcap_i(\Sigma\backslash Q_i) = \bigcap_i \{s\}\backslash(P;Q_i)$$

$$= \{s\}\backslash\bigcap_i(P;Q_i)$$

Case 2. $\{s\}\backslash P = \{t_0,t_1,\ldots,t_n\}$

$$\{s\}\backslash(P;\bigcap_i Q_i) = (\{s\}\backslash P)\backslash(\bigcap_i Q_i)$$

$$= \bigcup_{j\leq n}(\{t_j\}\backslash\bigcap_i Q_i)$$

$$= \bigcup_{j\leq n}(\bigcap_i \{t_j\}\backslash Q_i)$$

$$= \bigcap_i(\bigcup_{j\leq n}(\{t_j\}\backslash Q_i))$$

$$= \bigcap_i(\{s\}\backslash P)\backslash Q_i$$

$$= \{s\}\backslash\bigcap_i(P;Q_i)$$

Here we have used the set-theoretic fact that finite union distributes through the intersection of a descending chain.

From (7) we derive the corollary.

(9) If $\{P_i\}$ is a decreasing chain of total finitary relations, then

$$(\bigcap_i P_i)\backslash R = \bigcup_i (P_i\backslash R)$$

Proof:

$\overline{LHS} = \overline{(\bigcap_i P_i);R\backslash\overline{I}}\backslash\overline{I}$ corollary of theorem 6

$= \overline{\bigcap_i (P_i;(R\backslash\overline{I}))}\backslash\overline{I}$ 2.2 (7)

$= (\bigcup_i \overline{(P_i;(R\backslash\overline{I}))})\backslash\overline{I}$ set theory

$= \bigcap_i \overline{(P_i;(R\backslash\overline{I})}\backslash\overline{I})$ 1.1 (7) a

$= \bigcap_i \overline{(P_i\backslash R)}$ corollary of theorem 6

$= \overline{RHS}$ set theory

2.3 Weakest prespecifications and preconditions

This subsection gives the weakest prespecification for each of the constructs of the language $\mathcal{D}$. The variables P and Q are assumed to be relations in the language $\mathcal{D}$, and R is an arbitrary relation serving as a specification. But we start with two healthiness conditions

(1) $P \setminus 0 = 0$

Proof: See appendix theorem 18.

This corresponds to the healthiness condition

wp (P, false) = false

(2) If $R_i \subseteq R_{i+1}$, and $s\ R_i\ \bot \Longrightarrow (\{s\} \upharpoonright R_i = \Sigma)$ for all $i \geq 0$

$$P \setminus (\bigcup_{i \geq 0} R_i) = \bigcup_{i \geq 0} (P \setminus R_i)$$

Proof: Case 1 $\{s\} \upharpoonright P = \Sigma$

$$s_0\ (\bigcap_{i \geq 0} (\bar{R}_i ; \bar{P} \setminus \bar{1}))\ s \Longrightarrow \forall i.\ \{s_0\} \upharpoonright \bar{R}_i \neq \{\}$$

$$\equiv \forall i.\ \{s_0\} \upharpoonright \bar{R}_i \supseteq \{\bot\}$$

$$\Rightarrow \{s_0\} \upharpoonright (\bigcap_{i \geq 0} \bar{R}_i) \supseteq \{\bot\}$$

$$\Rightarrow s_0\ ((\bigcap_{i \geq 0} \bar{R}_i) ; \bar{P} \setminus \bar{1})\ s$$

Case 2. $\{s\} \upharpoonright P = \{t_0, t_1, \ldots, t_n\}$

$$s_0\ (\bigcap_{i \geq 0} (\bar{R}_i ; \bar{P} \setminus \bar{1}))\ s \Longrightarrow \forall i.\ (\{s_0\} \upharpoonright \bar{R}_i) \cap \{t_0, t_1, \ldots, t_n\} \neq \{\}$$

$$\Rightarrow \{s_0\} \upharpoonright (\bigcap_{i \geq 0} \bar{R}_i) \cap \{t_0, t_1, \ldots, t_n\} \neq \{\}$$

$$\Rightarrow s_0\ ((\bigcap_{i} \bar{R}_i) ; \bar{P} \setminus \bar{1})\ s$$

(3) $\underline{\text{abort}} \setminus R = \{s_0, s \mid \forall \dot{s}\ \ s_0\ R\ \dot{s}\}$

Proof: $U \setminus R = \{s_0, s \mid \forall \dot{s}.\ s\ U\ \dot{s} \Longrightarrow s_0\ R\ \dot{s}\}$

$= RHS.$

(4) $\underline{\text{skip}}\backslash R = k\backslash D \cap R \cup \bar{k}\backslash D \cap (U\backslash R)$

Proof. LHS $= (\bar{k} \cap I)^+\backslash R$ definition of skip

$= (\bar{k}\backslash D \cup I\backslash R) \cap (k\backslash D \cup U\backslash R)$ theorem 23

$= (k\backslash D \cap R) \cup (\bar{k}\backslash D \cap (U\backslash R))$

$\cup (k\backslash D \cap \bar{k}\backslash D) \cup (R \cap (U\backslash R))$ theorem 1

$=$ RHS theorem 13

Corollary

$\underline{\text{skip}}\backslash R = R$ provided $\bar{k}\backslash D \cap R = U\backslash R$

This corresponds to Dijkstra's

wp (skip,R) = R

(5) $(s := f(s))\backslash R = R;(\bar{f}\backslash \bar{I}) \cup (f\backslash D \cap U\backslash R)$

Proof. LHS $= f^+\backslash R$

$= R;(\bar{f}\backslash \bar{I}) \cup (f\backslash D \cap U\backslash R)$ corollary of theorem 25

If $U\backslash R = D$ or if f is a total function (from theorem 18 it follows that $f\backslash D = D$), then

$$(s := f(s))\backslash R = R;(\bar{f}\backslash \bar{I}) = \left\{ s_0, s \;\middle|\; s_0 R f(s) \right\}$$

This is analogous to Dijkstra's rule for assignment.

(6) $(P;Q)\backslash R = P\backslash(Q\backslash R)$

This rule holds for all relations.

(7) $\underline{\text{if}}\ b \longrightarrow P \,[]\, c \longrightarrow Q\ \underline{\text{fi}} \backslash R$

$= (b \backslash 0 \cup P \backslash R) \cap (c \backslash 0 \cup Q \backslash R)$

$\cap\ (\overline{b} \backslash 0 \cup \overline{c} \backslash 0 \cup U \backslash R)$

Proof: LHS $= (b \cap P \cup c \cap Q)^{+} \backslash R$ definition of alternative command

$=$ RHS theorem 23

If $U \backslash R = 0$, this is similar to Dijketra's law

$$wp\ (\underline{\text{if}}\ b \longrightarrow P \,[]\, c \longrightarrow Q\ \underline{\text{fi}}, R) =$$

$$(b \longrightarrow wp\ (P,R)) \cap (c \longrightarrow wp\ (Q,R)) \wedge (b \vee c)$$

(8) $\underline{\text{do}}\ b \longrightarrow P\ \underline{\text{od}} \backslash R = \bigcup_{n \geq 0} H_n$

where $H_0 = U \backslash R$

and $H_{n+1} = (b \backslash 0 \cup P \backslash H_n) \cap (\neg b \backslash 0 \cup \text{skip} \backslash R) \cap (\overline{b \backslash 0} \cup \overline{\neg b \backslash 0} \cup U \backslash R)$

Proof: Let $F(X) = \underline{\text{if}}\ b \longrightarrow (P;X) \,[]\, \neg b \longrightarrow \underline{\text{skip}}\ \underline{\text{fi}}$

LHS $= (\bigcap_{n \geq 0} F^n(U)) \backslash R$

$= \bigcup_{n \geq 0} (F^n(U) \backslash R)$ by 2.2 (9)

We therefore need to prove by induction

$$F^n(U)\backslash R = H_n \qquad \text{for all } n.$$

(0) $F^0(U)\backslash R = U\backslash R = H_0$

(1) $F^{n+1}(U)\backslash R = F(F^n(U))\backslash R$

$$= \underline{\text{if}}(b \longrightarrow P;F^n(U) \;[]\; \neg b \longrightarrow \underline{\text{skip}}\ \underline{\text{fi}})\backslash R$$

$$= (b\backslash 0 \cup P\backslash(F^n(U)\backslash R)) \cap (\neg b\backslash 0 \cup (\underline{\text{skip}}\backslash R))$$

$$\cap (\overline{b}\backslash 0 \cup \overline{\neg b}\backslash 0 \cup U\backslash R) \qquad \text{by 2.3 (7)}$$

$$= (b\backslash 0 \cup P\backslash H_n) \cap (\neg b\backslash 0 \cup (\underline{\text{skip}}\backslash R))$$

$$\cap (\overline{b\backslash 0} \cup \overline{\neg b\backslash 0} \cup U\backslash R)$$

theorem 13 and induction hypothesis

$$= H_{n+1}$$

3. The VDM connection

In specifying the desired behaviour of a sequential program, it is quite natural to describe this as a relation R between the initial values of the variables on entry to the program and the final values of the variables on successful termination of the program. it is unnecessary to make any mention of the possibility of non-termination, because it can be taken for granted that we never wish to specify that a program must fail to terminate. It therefore seems unnecessary, as well as unnatural, to introduce the fictitious non-terminated state $\perp$ into the mathematics of the language $\mathfrak{D}$, even though it greatly simplifies the proofs. In this section we describe how this problem is solved in the Vienna Development Method [3].

Let the specification R be a relation between real initial and final states (i.e. excluding $\perp$). Suppose we know that the program specified by R will never be entered unless the initial values of the variables satisfy a certain condition b. The designer of the program can rely on the assumption that b holds on entry; it is the responsibility of some other component of the system to guarantee the truth of this assumption. Thus a programming task is specified in VDM as a pair

$$(b,R)$$

where b is a condition describing only the initial state and R is a relation between initial and final states. The relation R in VDM is misleadingly called a postcondition.

The VDM specification (b,R) can be readily translated to a specification for the programming language $\mathfrak{D}$, in which $\perp$ is introduced to represent non-termination. Suppose that a program P

meets this specification, but the environment fails to meet its part of the obligation, and the program is in fact entered in an initial state which fails to meet the condition b. In the event of such a violation, the program P has had its basic assumption falsified; and consequently it might do anything whatsoever - it may even fail to terminate. The specifier, having promised that b will be true, has no right to place any restriction on the behaviour of the program when the promise is not kept. We therefore define

$$(b,R) = \bar{b} \cup R$$

where the complement $\bar{b}$ is relative to a Σ which includes $\perp$. A relation of the form given in this definition is said to be expressible in VDM.

This definition of the VDM specification shows a very natural and convenient way of specifying programs that in certain circumstances are allowed to fail; and it does so without explicit mention of the awkward fictitious state $\perp$. In the remainder of this section we will show that all specifications can for all practical purposes be expressed in the VDM format. (The qualifications of the previous sentence are necessary, because relations like $\{(\perp,\perp)\}$ cannot be so expressed.) The first task will be to show that

(1) All programs of $\mathfrak{D}$ can be expressed in VDM.

For all practical purposes two specifications are <u>equivalent</u> if they are satisfied by exactly the same set of programs of the language $\mathfrak{D}$. Thus our second task is to prove that

(2) For any relation R, there is an equivalent relation expressible in VDM.

3.1 The language $\mathcal{D}$ defined in VDM

The first task is to show that every program of the language $\mathcal{D}$ can be expressed in the VDM format

$$(b,R) = \overline{b} \cup R$$

where b is a condition and R is a relation, and neither of them mention $\perp$. This is done by defining each construct of the language $\mathcal{D}$ in terms of VDM, assuming that each operand of the construct is also expressed in VDM. The desired result follows by structural induction on the program expressed in $\mathcal{D}$.

(1) $U = (0,R)$

(where it does not matter what R is).

Proof: $(0,R) = \overline{0} \cup R$ by definition of VDM

$= U$

(2) If P is a partial function (with $\perp$ not in its domain or range) then

$$P^+ = ((P;U),P)$$

Proof: $P^+ = P \cup \overline{P;U}$ definition of $^+$

$= (P;U,P)$ definition of VDM

(3) $(b,P);(c,Q) = (b \cap \overline{P;\overline{c}}, P;Q)$

Proof: LHS $= (\overline{b} \cup P) ; (\overline{c} \cup Q)$

$= \overline{b} ; (\overline{c} \cup Q) \cup P;\overline{c} \cup P;Q$

$= \overline{b} \cup P;\overline{c} \cup P;Q$ since $\{\perp\} \times \Sigma \subseteq (\overline{c} \cup Q)$

$= \overline{(b \cap \overline{P;\overline{c}})} \cup P;Q$ set theory

$=$ RHS

Furthermore we have

$$(P;\overline{c});U = P;(\overline{c};U)$$
$$= P;\overline{c} \qquad \text{since } \overline{c} \text{ is a condition}$$

thus $P;\overline{c}$ and its negation $\overline{P;\overline{c}}$ are also conditions.

(4) $(b \wedge (d,P) \vee c \wedge (e,Q))^+ = ((b \vee c) \wedge (\overline{b} \vee d) \wedge (\overline{c} \vee e),\ b \wedge P \vee c \wedge Q)$

Proof: LHS $= b \wedge (\overline{d} \vee P) \vee c \wedge (\overline{e} \vee Q) \vee \overline{(b \vee c)}$ corollary of theorem 21

$= \overline{\overline{(b \vee c) \wedge (\overline{b \wedge \overline{d}}) \wedge (\overline{c \wedge \overline{e}})}} \vee (b \wedge P) \vee (c \wedge Q)$ set theory

$=$ RHS definition of VDM

(5) If $(b_i, P_i) \in \mathfrak{D}$. and $(b_{i+1}, P_{i+1}) \subseteq (b_i, P_i)$ for all $i \geq 0$

then $\bigcap_i (b_i, P_i) = (\bigcup_i b_i, \bigcup_{i \geq 0} \bigcap_{j \geq i} (b_j \wedge P_j))$

Lemma. If $(b_{i+1}, P_{i+1}) \subseteq (b_i, P_i)$ for all $i \geq 0$, then

$$\overline{b}_{i+1} \subseteq \overline{b}_i$$

and $b_i \wedge P_{i+1} \subseteq b_i \wedge P_i$ for all $i \geq 0$

Proof: $\overline{b}_{i+1} = \overline{b}_{i+1};(\{\bot\} \times \Sigma)$ since $\overline{b}_{i+1}$ is a condition

$= (\overline{b}_{i+1} \wedge (\Sigma \times \{\bot\}));U$ theorem 14

$= ((\overline{b}_{i+1} \vee P_{i+1}) \wedge (\Sigma \times \{\bot\}));U$ since P_{i+1} doesn't mention $\bot$

$\subseteq ((\overline{b}_i \vee P_i) \wedge (\Sigma \times \{\bot\}));U$ by the assumption

$= \overline{b}_i$

$$b_i \cap P_{i+1} = b_i \cap (\overline{b}_{i+1} \cup P_{i+1}) \qquad \text{since } \overline{b}_{i+1} \subseteq \overline{b}_i$$

$$\subseteq b_i \cap (\overline{b}_i \cup P_1) \qquad \text{by the assumption}$$

$$= b_i \cap P_i$$

Proof of theorem

Case 1. $(s_0,s_1) \in \overline{b}_i$ for all $i \geq 0$

It is obvious that

$$(s_0,s_1) \in \bigcap_{i \geq 0} \overline{b}_i \subseteq \text{LHS} \cap \text{RHS}$$

Case 2. There exists K such that $(s_0,s_1) \overline{\in} \overline{b}_K$

Suppose that $(s_0,s_1) \in \bigcap_{i \geq 0} (b_i,P_i)$ From lemma it follows that

$$(s_0,s_1) \in b_i \cap P_1 \qquad \text{for all } i \geq K$$

which implies that

$$(s_0,s_1) \in \bigcap_{i \geq K} (b_i \cap P_i) \subseteq \text{RHS}$$

Now assume that $(s_0,s_1) \in (\bigcup_{K \geq 0} b_K, \bigcup_{K \geq 0} \bigcap_{i \geq K} (b_i \cap P_i))$. There exists n such that

$$(s_0,s_1) \in b_i \cap P_i \qquad \text{for all } i \geq n.$$

When n = 0, we obtain

$$(s_0,s_1) \in \bigcap_{i \geq 0} (b_i \cap P_i) \subseteq \text{LHS}$$

Otherwise there are two subcases

Case 2.1 $(s_0,s_1) \overline{\in} b_{n-1}$

$$(s_0,s_1) \in \overline{b}_{n-1} \subseteq \overline{b}_{n-2} \subseteq \cdots \subseteq \overline{b}_0$$

which induces to

$$(s_0,s_1) \in \bigcap_{i \leq n-1} \overline{b}_i \cap \bigcap_{i \geq n} (b_i \cap P_i) \subseteq \text{LHS}$$

Case 2.2 $(s_0,s_1) \in b_{n-1}$

$$(s_0,s_1) \in b_{n-1} \cap (\dot{b}_n \cap P_n) \subseteq b_{n-1} \cap P_{n-1}$$

Then we repeat the same argument for n-1 as above for n. Finally we can show $(s_0,s_1) \in$ LHS also.

Combine these cases we conclude that LHS = RHS.

These five theorems show that the programming language $\mathcal{D}$ can be wholly defined within the VDM notation. Furthermore, if (b,P) is expressed as a program in $\mathcal{D}$, it has the additional important property that every state satisfying the condition b is within the domain of P, or more formally

(6) $b \subseteq P;U$

Proof: since (b,P) is a total relation

$$(\bar{b} \vee P);U = U$$

but LHS $= \bar{b};U \vee P;U$

$= \bar{b} \vee P;U$ since $\bar{b}$ is a condition

The conclusion follows by set theory.

It remains to define the weakest prespecification of a pair of specifications expressed in the VDM style.

(7) $(b,Q)\backslash(c,R) = (\overline{(c,R)};U,\overline{b\backslash 0} \cap (Q\backslash(c,R)))$

Proof: LHS $= (\bar{b}\backslash(c,R)) \cap (Q\backslash(c,R))$ 1.1(7)

$= (\overline{b\backslash 0} \cup (U\backslash(\acute{c},R))) \cap (Q\backslash(\acute{c},R))$ theorem 15

$= (\overline{b\backslash 0} \cap (Q\backslash(c,R))) \cup ((U\backslash(c,R)) \cap (Q\backslash(\acute{c},R)))$ set theory

$= (\overline{b\backslash 0} \cap (Q\backslash(c,R))) \cup (U\backslash(c,R))$ 1.1(7)

$=$ RHS corollary 4 of theorem 13

3.2 All specifications can be equivalently expressed in VDM

Let R and S be two relations, intended to be used as specifications of a program in the language $\mathcal{D}$. Suppose also that *every program* in $\mathcal{D}$ which satisfies R also satisfies S, and vice versa. Then it cannot matter which of the relations R and S is taken as the original specification; each of them gives exactly the same range of choices to the designer of the program. The two specifications are therefore said to be equivalent.

Definition. R and S are <u>equivalent</u> if

$$\forall p \in \mathcal{D} \quad p \subseteq R \equiv p \subseteq S$$

Corollary: all unimplementable specifications are equivalent.

Clearly this *concept of equivalence* is dependent on the definition of the programming language, in this case $\mathcal{D}$. For other languages, R and S may indeed specify different programs.

We can now show that the VDM style of specification is adequate for all purposes in the specification and development of programs in $\mathcal{D}$, and there is no need ever to make explicit mention of the fictitious state $\perp$. This state has been extremely useful in exploring the mathematical properties of the language $\mathcal{D}$; but in the application of the mathematics, the VDM style offers a convenient *way of avoiding* its explicit mention.

Theorem. For any specification S, there is an equivalent specification (b,R)

Proof: *define*

$$\mathcal{D}(S) = \{(c,p) \mid (c,p) \in \mathcal{D} \wedge (c,p) \subseteq S\}$$

$$b = \bigcap \{c \mid (c,p) \in \mathcal{D}(S)\}$$

$$R = \bigcup \{p \mid (c,p) \in \mathcal{D}(S)\}$$

Now we have

$$(b,R) \quad = \quad \bigcup \left\{ (c,P) \;\middle|\; (c,P) \in \mathcal{D}(S) \right\} \subseteq S$$

On the other hand if $P \in \mathcal{D}$ and $P \subseteq S$, then we know that $P \in \mathcal{D}(S)$ which implies that $P \subseteq (b,R)$

This completes the proof.

Acknowledgement

The inspiration of J.-R. Abrial [1] and E.W. Dijkstra [2] has been pervasive in our work. We are grateful for helpful comments from members of the Programming Research Group; and particularly Dr. Ian Hayes. Work on this paper was supported in part by SERC grant no. GR/B/55435.

References

1. J.-R. Abrial

The Mathematical Construction of a Program.
Science of Computer Programming 4 (1984).

2. E.W. Dijkstra

Guarded commands, nondeterminacy and the formal derivation of programs.
Comm. of the ACM 18 (1975), 453-457.

3. C. Jones

Software Development: A Rigorous Approach.
Prentice Hall (1980).

4. B. Sufrin

Mathematics for System Specification.
Lecture Notes 1983/1984.
Programming Research Group, Oxford (1983).

5. A. Tarski

A Lattice-Theoretical Fixpoint Theorem and its Applications.
Bull. Amer. Math. Soc. 55 (1949), 1051-1052 and 1192.

6. A. Tarski

On the Calculus of Relations.
J. Symb. Logic, 6 (1941), 73-89.

Appendix

This appendix develops the relational calculus using the weakest prespecification (as defined in 1.1 (10)) in place of the more familiar converse operator. All proofs are based on just two axioms (in addition to two familiar axioms for ;)

1 $P;Q \subseteq R \;\equiv\; P \subseteq Q \backslash R$

Proof: see 1.1 (10)

2 $\overline{Q \backslash R} = (\bar{R} \backslash \bar{Q}) \backslash \bar{I}$

Proof: see 1.1 (13)

These two axioms are sufficient to prove the three axioms of Tarski [6] which relate to converse, namely

$\breve{P} = P$ (Theorem 4 below)

$\widetilde{P;Q} = \breve{Q};\breve{P}$ (Theorem 6 below)

$(P;Q) \cap \dot{R} = 0 \;\equiv\; (Q;R) \cap \breve{P} = 0$ (Theorem 11 below)

Replacement of this last axiom will not be lamented.

Definitions

1. For any relation R, its converse $\breve{R}$ is defined as the weakest prespecification of $\bar{R}$ with respect to $\bar{I}$, i.e.

$$\breve{R} = \bar{R} \backslash \bar{I}$$

2. Let P and R be relations. The weakest postspecification of P with respect to R, denoted by R/P, is defined as the weakest prespecification of the converse of $\bar{R}$ with respect to the converse of $\bar{P}$, i.e.

$$R/P = (R \backslash \bar{I}) \backslash (P \backslash \bar{I})$$

Theorem 1

The identity relation gives no help in meeting any specification

$$P = I \backslash P$$

Proof: $X \subseteq P \equiv X;I \subseteq P$

$\equiv X \subseteq I \backslash P$ axiom 1

Theorem 2

In order to meet specification R with the aid of (P;Q), you must ensure that P can meet R with the aid of Q

$$(P;Q) \backslash R = P \backslash (Q \backslash R)$$

Proof: $X \subseteq (P;Q) \backslash R \equiv X;P;Q \subseteq R$ axiom 1

$\equiv X;P \subseteq Q \backslash R$ axiom 1

$\equiv X \subseteq P \backslash (Q \backslash R)$ axiom 1

Theorem 3

The operations complement and converse commute

$$\overline{\breve{P} \backslash \overline{I}} = P \backslash \overline{I} \qquad \text{i.e., } \breve{\overline{P}} = \overline{\breve{P}}$$

Proof: LHS $= (I \backslash P) \backslash \overline{I}$ axiom 2

$=$ RHS theorem 1

Theorem 4

The converse operation is the inverse of itself

$$(P \backslash \overline{I}) \backslash \overline{I} = P \qquad \text{i.e., } \breve{\breve{P}} = P$$

Proof: LHS $= \overline{\overline{I} \backslash \overline{\overline{P}}}$ axiom 2

$= \overline{\overline{P}}$ theorem 1

$= P$

Theorem 5

$$(P \backslash Q) \backslash \overline{I} = P;(Q \backslash \overline{I})$$

Proof: LHS $= (P \backslash ((Q \backslash \overline{I}) \backslash \overline{I})) \backslash \overline{I}$ theorem 4

$= ((P;(Q \backslash \overline{I})) \backslash \overline{I}) \backslash \overline{I}$ theorem 2

$= P;(Q \backslash \overline{I})$ theorem 4

Corollary

$$\overline{P \backslash R} = \breve{R};(\breve{P} \backslash \overline{I})$$

Proof: LHS $= (\overline{R} \backslash \overline{P}) \backslash \overline{I}$ axiom 2

$= \overline{R};(\overline{P} \backslash \overline{I})$ theorem 5

Theorem 6

The converse of P composed with Q is equal to the converse of Q composed with the converse of P.

$$\overline{P;Q} \backslash \overline{I} = (\overline{Q} \backslash \overline{I});(\overline{P} \backslash \overline{I})$$ i.e. $\breve{P;Q} = \breve{Q};\breve{P}$

Proof: LHS $= \overline{(P;Q) \backslash \overline{I}}$ theorem 3

$= \overline{(P \backslash (Q \backslash \overline{I}))}$ theorem 2

$= ((\overline{Q \backslash \overline{I}}) \backslash \overline{P}) \backslash \overline{I}$ axiom 2

$= (\overline{Q} \backslash \overline{I});(\overline{P} \backslash \overline{I})$ theorem 3 and theorem 5

Corollary

$$\overline{P \backslash R} = (\overline{\overline{P;R} \backslash \overline{I}}) \backslash \overline{I}$$

Proof: RHS $= ((\overline{R} \backslash \overline{I}) \backslash \overline{I});(\overline{P} \backslash \overline{I})$ theorem 6

$= \overline{R};(\overline{P} \backslash \overline{I})$ theorem 4

$= \overline{P \backslash R}$ corollary of theorem 5

Theorem 7

The converse operation is monotonic.

$$P \subseteq Q \longrightarrow (\overline{P} \backslash \overline{I}) \subseteq (\overline{Q} \backslash \overline{I})$$

Proof: $X \subseteq \overline{P} \backslash \overline{I} \equiv X ; \overline{P} \subseteq \overline{I}$ axiom 1

$\Rightarrow X ; \overline{Q} \subseteq \overline{I}$ since $\overline{P} \supseteq \overline{Q}$

$\equiv X \subseteq \overline{Q} \backslash \overline{I}$ axiom 1

Corollary

$$(\overline{P} \backslash \overline{I}) \subseteq (\overline{Q} \backslash \overline{I}) \equiv P \subseteq Q$$

Proof: direct from theorem 4 and theorem 7.

Theorem 8

P composed with Q will meet the specification R iff Q meets the weakest postspecification of P with respect to R.

$$P ; Q \subseteq R \equiv Q \subseteq (R \backslash \overline{I}) \backslash (P \backslash \overline{I})$$

Proof: LHS $\equiv (\overline{P ; Q}) \backslash \overline{I} \subseteq (\overline{R} \backslash \overline{I})$ theorem 7 and corollary

$\equiv (\overline{Q} \backslash \overline{I}) ; (\overline{P} \backslash \overline{I}) \subseteq (\overline{R} \backslash \overline{I})$ theorem 6

$\equiv (\overline{Q} \backslash \overline{I}) \subseteq (\overline{P} \backslash \overline{I}) \backslash (\overline{R} \backslash \overline{I})$ axiom 1

$\equiv (\overline{Q} \backslash \overline{I}) \subseteq \overline{((\overline{R} \backslash \overline{I}) \backslash (\overline{P} \backslash \overline{I}))} \backslash \overline{I}$ axiom 2

$\equiv (\overline{Q} \backslash \overline{I}) \subseteq \overline{((R \backslash \overline{I}) \backslash (P \backslash \overline{I}))} \backslash \overline{I}$ theorem 3

$\equiv Q \subseteq (R \backslash \overline{I}) \backslash (P \backslash \overline{I})$ theorem 7 and corollary

Theorem 9

The identity relation is the weakest prespecification of its complement with respect to itself.

$$\overline{I} \backslash \overline{I} = I$$

Proof: LHS $= I;(\overline{I}\backslash\overline{I})$

$= (I\backslash\overline{I})\backslash\overline{I}$ theorem 5

$= 1$ theorem 4

Theorem 10

P composed with Q meets $\overline{I}$ iff so does Q composed with P. (The discovery of this simple theorem about the fixed points of relations astonished each of the authors individually.)

$$P;Q \subseteq \overline{I} \equiv Q;P \subseteq \overline{I}$$

Proof: LHS $\equiv Q \subseteq (\overline{I}\backslash\overline{I})\backslash(P\backslash\overline{I})$ theorem 8

$\equiv Q \subseteq 1\backslash(P\backslash\overline{I})$ theorem 9

$\equiv Q \subseteq (P\backslash\overline{I})$ theorem 1

$\equiv Q;P \subseteq \overline{I}$ axiom 1

Corollary $P\backslash\overline{I} = \overline{I}/P$

Proof: $X \subseteq P\backslash\overline{I} \equiv X;P \subseteq \overline{I}$ axiom 1

$\equiv P;X \subseteq \overline{I}$ theorem 10

$\equiv X \subseteq \overline{I}/P$ theorem 8

Now we come to show one of the basic theorems in the calculus of relations defined by A. Tarski [6]

Theorem 11

$$(P;Q)\cap(\overline{R}\backslash\overline{I}) = 0 \equiv (Q;R)\cap(\overline{P}\backslash\overline{I}) = 0$$

Proof: LHS $\equiv$ $(P;Q)\subseteq \overline{\overline{R}\backslash\overline{I}}$ set theory

$\equiv$ $(P;Q)\subseteq R\backslash\overline{I}$ theorem 3

$\equiv$ $(P;Q);R\subseteq\overline{I}$ axiom 1

$\equiv$ $Q;R\subseteq\overline{I}/P$ theorem 8

$\equiv$ $Q;R\subseteq P\backslash\overline{I}$ corollary of theorem 10

$\equiv$ RHS theorem 3

Definition

A relation b is said to be a condition if

$$b = b;U$$

i.e. it relates everything in its domain to anything whatsoever.

Theorem 12

1. b is a condition iff its converse $\overline{b}\backslash\overline{I}$ relates everything to to anything in the domain of b.

$$b = b;U \equiv \overline{b}\backslash\overline{I} = U;(\overline{b}\backslash\overline{I})$$

2. If b is a condition so its complement $\overline{b}$

$$b = b;U \Longrightarrow \overline{b} = \overline{b};U$$

Proof: 1. LHS $\equiv$ $\overline{b}\backslash\overline{I} = \overline{b;U}\backslash\overline{I}$ theorem 7

$\equiv$ $\overline{b}\backslash\overline{I} = (O\backslash\overline{I});(\overline{b}\backslash\overline{I})$ theorem 6

$\equiv$ $\overline{b}\backslash\overline{I} = U;(\overline{b}\backslash\overline{I})$ since $U;O = O\subseteq\overline{I}$

2. LHS $\Longrightarrow$ $b;U\cap\overline{b} = O$ set theory

$\equiv$ $b;U\cap(\overline{b}\backslash\overline{I})\backslash\overline{I} = O$ theorem 4

$\equiv$ $U;\overline{\overline{b}\backslash\overline{I}}\cap\overline{b}\backslash\overline{I} = O$ theorem 11

$\Longrightarrow$ $U;b\backslash\overline{I}\subseteq b\backslash\overline{I}$ theorem 3

$\equiv$ $U;b\backslash\overline{I} = b\backslash\overline{I}$ since $U;R\supseteq R$

$\equiv$ $\overline{b} = \overline{b};U$ theorem 12.1

Theorem 13

If b is a condition, then its converse is given by

1. $\overline{b}\backslash\overline{I} = \overline{b\backslash 0}$

and 2. $\overline{b}\backslash\overline{I} = \overline{b}\backslash 0$

Proof: 1. LHS $= U;\overline{b}\backslash\overline{I}$ — theorem 12

$= \overline{b\backslash 0}$ — corollary of theorem 5

2. LHS $= \overline{\overline{\overline{b}\backslash\overline{I}}}$ — theorem 3

$= \overline{\overline{\overline{b}\backslash 0}}$ — theorem 12 and theorem 13.1

$= \overline{b}\backslash 0$

Corollary

1. $U\backslash\overline{I} = U\backslash 0 = 0$
2. $(P;U)\backslash\overline{I} = P\backslash 0$
3. $0\backslash\overline{I} = 0\backslash 0 = U$
4. $\overline{U\backslash P} = \overline{P};U$

Proof: 1. LHS $= U\backslash 0$ — theorem 13

$= 0$ — since $X;U\subseteq 0 \Rightarrow X\subseteq 0$

2. LHS $= P\backslash(U\backslash\overline{I})$ — theorem 2

$= P\backslash 0$ — corollary 1 of theorem 13

3. $0\backslash\overline{I} = 0\backslash 0$ — set P = 0 in corollary 2 of theorem 13

$= \overline{U\backslash 0}$ — theorem 13

$= \overline{0}$ — corollary 1 of theorem 13

$= U$

4. LHS $= \overline{P};0\backslash\overline{I}$ — corollary of theorem 5

$= \overline{P};U$ — corollary 3 of theorem 13

In the following part we shall use lower case letters for conditions.

Theorem 14

$$P;(b \wedge Q) = (P \wedge \overline{b \backslash 0});Q \qquad \text{i.e., } P;(b \wedge Q) = (P \wedge \breve{b});Q$$

On the left hand side, Q has its domain restricted to the condition b. On the right hand side, P has its codomain subject to the same restriction. When P and Q are composed, it does not matter whether the restriction takes place after P or before Q.

Proof: LHS $= (P \wedge (b \backslash 0));(b \wedge Q) \vee (P \wedge (\overline{b \backslash 0}));(b \wedge Q)$

$= (P \wedge (\overline{b \backslash 0}));(b \wedge Q)$ since $(b \backslash 0);b \leq 0$

$= (P \wedge (\overline{b \backslash 0}));(b \wedge Q) \vee (P \wedge (\overline{\bar{b} \backslash 0}));(\bar{b} \wedge Q)$ since $(\bar{b} \backslash 0);\bar{b} \leq 0$

$= (P \wedge (\overline{b \backslash 0}));(b \wedge Q) \vee (P \wedge (\overline{b \backslash 0}));(\bar{b} \wedge Q)$ theorem 13

$=$ RHS

Corollary

$$X;((P;U) \wedge Q) = (X \wedge (\overline{P \backslash 0}));Q$$

Proof: LHS $= (X \wedge \overline{(P;U) \backslash 0});Q$ theorem 14

$= (X \wedge \overline{P \backslash (U \backslash 0)});Q$ theorem 2

$= (X \wedge \overline{P \backslash 0});Q$ corollary 1 of theorem 13

Theorem 15

Prespecification distributes through conjunction in its first argument if one of the limbs of the conjunction is of the form P;U

$$((P;U) \wedge Q) \backslash R = P \backslash 0 \vee Q \backslash R$$

Proof: $X \subseteq LHS \equiv X;((P;U) \cap Q) \subseteq R$ axiom 1

$\equiv (X \cap (\overline{P \backslash 0})); Q \subseteq R$ corollary of theorem 14

$\equiv X \cap (\overline{P \backslash 0}) \subseteq Q \backslash R$ axiom 1

$\equiv X \subseteq P \backslash 0 \cup Q \backslash R$ set theory

Corollary

$$(b \cap Q) \backslash R = b \backslash 0 \cup Q \backslash R$$

Proof: LHS $= ((b;U) \cap Q) \backslash R$ since $b = b;U$

$= b \backslash 0 \cup Q \backslash R$ theorem 15

Theorem 16

Prespecification distributes through disjunction in its first argument

$$(P \cup Q) \backslash R = (P \backslash R) \cap (Q \backslash R)$$

Proof: $X \subseteq (P \cup Q) \backslash R \equiv X;(P \cup Q) \subseteq R$ axiom 1

$\equiv (X;P \subseteq R) \wedge (X;Q \subseteq R)$ set theory

$\equiv (X \subseteq P \backslash R) \wedge (X \subseteq Q \backslash R)$ axiom 1

$\equiv X \subseteq (P \backslash R) \cap (Q \backslash R)$ set theory

Theorem 17

Prespecification also distributes through conjunction in its second argument

$$Q \backslash (R \cap S) = (Q \backslash R) \cap (Q \backslash S)$$

Proof: Similar to theorem 16.

Corollary

$$Q \backslash 0 = 0 \Longrightarrow Q \backslash R \subseteq \overline{Q \backslash \overline{R}}$$

Proof:

$(Q \backslash R) \cap (Q \backslash \overline{R})$	$= Q \backslash (R \cap \overline{R})$	theorem 17
	$= Q \backslash 0$	set theory
	$= 0$	by the assumption

from which the result follows.

Definitions

1. P is a total relation if it has universal domain

$$P;U = U$$

2. A relation P is said to be a total function if

$$\overline{P} = P;\overline{I}$$

Theorem 18

If P is a total relation, then its weakest prespecification with respect to nuil relation is null.

$$P;U = U \Longrightarrow P \backslash 0 = 0$$

Proof:

$P \backslash 0$	$= (P;U) \backslash \overline{I}$	corollary 2 of theorem 13
	$= U \backslash \overline{I}$	by the assumption
	$= 0$	corollary 1 of theorem 13

Corollary

If P is a total function then

1. $P \backslash D = 0$
2. $\overline{P} \backslash \overline{I} = P \backslash I$

Proof: 1. $P;U = P;(I \cup \overline{I})$ set theory

$= P;I \cup P;\overline{I}$ the distributive law of ;

$= P \cup \overline{P}$ by the assumption

$= U$

from which and theorem 18 we reach the conclusion

2. LHS $= (P;\overline{I}) \backslash \overline{I}$ by the assumption

$= P \backslash (\overline{I} \backslash \overline{I})$ theorem 2

$= P \backslash I$ theorem 9

Theorem 19

When P is a total function, its weakest prespecification is the sequential composition of the given specification R and the converse of P.

$$P \backslash R = R;(P \backslash I)$$

Proof: $RHS;P = R;((P \backslash I);P)$ the associative law of ;

$\subseteq R;I$ axiom 1

$= R$

$\therefore RHS \subseteq P \backslash R$

On the other hand we also have

$P \backslash R \subseteq \overline{\overline{P} \backslash \overline{R}}$ corollary of theorem 17

$= R;(\overline{P} \backslash \overline{I})$ corollary of theorem 5

$= R;(P \backslash I)$ corollary 2 of theorem 18

Definition

For any relation P, $P;U$ is a condition which holds just for all initial states in the domain of P. We define P^+ as a *relation* that behaves like P if started in the domain of P, but does anything whatsoever if started outside that domain

$$P^+ = P \cup \overline{P;U}$$

Theorem 20

P^+ is a total relation

Proof: $P^+;U = (P;U) \cup (\overline{P;U};U)$ Definition of +

$= (P;U) \cup (\overline{P;U})$ theorem 12.2

$= U$

Theorem 21

$$\overline{((b \cap P) \cup (c \cap Q));U} = (\overline{b} \cup \overline{P;U}) \cap (\overline{c} \cup \overline{Q;U})$$

Proof: LHS $= \overline{(b \cap P);U \cup (c \cap Q);U}$ the distributive law of ;

$= \overline{(I \cap b\backslash 0);P;U \cup (I \cap c\backslash 0);Q;U}$ theorem 14

$= \overline{(b \cap P;U) \cup (c \cap Q;U)}$ theorem 14

$= \overline{(b \cap P;U)} \cap \overline{(c \cap Q;U)}$ set theory

$= (\overline{b} \cup \overline{P;U}) \cap (\overline{c} \cup \overline{Q;U})$ set theory

Corollary

If P and Q both are total relations, then

$$\overline{((b \cap P) \cup (c \cap Q));U} = \overline{b} \cap \overline{c}$$

Proof: LHS $= (\overline{b} \cup \overline{P;U}) \cap (\overline{c} \cup \overline{Q;U})$ theorem 21

$= \overline{b} \cap \overline{c}$ since $P;U = Q;U = U$

Lemma 1

$$\overline{P;U}\backslash R = (\overline{P\backslash 0}) \cup (U\backslash R)$$

Proof:

LHS	=	$(\overline{P;U} \cap U)\backslash R$	set theory
	=	$(\overline{P;U}\backslash 0) \cup (U\backslash R)$	theorem 15
	=	$(\overline{P;U\backslash 0}) \cup (U\backslash R)$	theorem 13
	=	$(\overline{P\backslash(U\backslash 0)}) \cup (U\backslash R)$	theorem 2
	=	$(\overline{P\backslash 0}) \cup (U\backslash R)$	corollary 1 of theorem 13

Theorem 22

$$P^{+}\backslash R = P\backslash R \cap (\overline{P\backslash 0} \cup U\backslash R)$$

Proof:

LHS	=	$(P \cup \overline{P;U})\backslash R$	definition of +
	=	$(P\backslash R) \cap (\overline{P;U}\backslash R)$	theorem 17
	=	$P\backslash R \cap (\overline{P\backslash 0} \cup U\backslash R)$	lemma 1

Lemma 2

If P and Q both are total relations, than

$$\overline{(b \cap P \cup \bar{c} \cap Q);U}\backslash R = \bar{b}\backslash 0 \cup \bar{c}\backslash 0 \cup U\backslash R$$

Proof:

LHS	=	$(\bar{b} \cap \bar{c})\backslash R$	corollary of theorem 21
	=	$\overline{(b \cup c)\backslash 0} \cup U\backslash R$	lemma 1
	=	$\bar{b}\backslash 0 \cup \bar{c}\backslash 0 \cup U\backslash R$	theorem 16

Theorem 23

If both P and Q are total relations, then

$$(b \cap P \cup c \cap Q)^+ \backslash R = (b \backslash 0 \cup P \backslash R) \cap (c \backslash 0 \cup Q \backslash R) \cap (\overline{b} \backslash 0 \cup \overline{c} \backslash 0 \cup U \backslash R)$$

Proof:

LHS	$= (b \cap P) \backslash R \cap (c \cap Q) \backslash R \cap \overline{(b \cap P \cup c \cap Q);U} \backslash R$	theorem 17 and 16
	$= (b \backslash 0 \cup P \backslash R) \cap (c \backslash 0 \cup Q \backslash R) \cap \overline{(b \cap P \cup c \cap Q);U} \backslash R$	corollary of theorem 15
	$=$ RHS	lemma 2

Lemma 3

$$(P;U) \cap I \subseteq P;(\overline{P} \backslash \overline{I}) \qquad \text{i.e. } (P;U) \cap I \subseteq P;\breve{P}$$

On the left hand side, the domain of identity relation is restricted to the domain of P. The same restriction is not quite so severe on the right hand side.

Proof:

LHS	$= (P;(P \backslash \overline{I} \cup \overline{P \backslash \overline{I}})) \cap I$	set theory
	$= (P;(P \backslash \overline{I})) \cap I \cup (P;\overline{P \backslash \overline{I}}) \cap I$	set theory
	$\subseteq \overline{I} \cap I \cup (P;(\overline{P} \backslash \overline{I})) \cap I$	axiom 1 and theorem 10
	$\subseteq P;(\overline{P} \backslash \overline{I})$	set theory

Theorem 24

$$P \backslash R \subseteq R;(\overline{P} \backslash \overline{I}) \cup P \backslash 0$$

This theorem places a useful upper bound on the weakest prespecification.

Proof: $X \subseteq P \backslash R \equiv X;P \subseteq R$ axiom 1

$\Longrightarrow X;P;(\overline{P}\backslash\overline{I}) \subseteq R;(\overline{P}\backslash\overline{I})$ by the monotonicity of ;

$\Longrightarrow X;(P;U \cap I) \subseteq R;(\overline{P}\backslash\overline{I})$ lemma 3

$\equiv X \cap \overline{P \backslash 0} \subseteq R;(\overline{P}\backslash\overline{I})$ theorem 14

$\Longrightarrow X \subseteq R;(\overline{P}\backslash\overline{I}) \cup P \backslash 0$ set theory

Definition

P is said to be a partial function if

$$P;\overline{I} \subseteq \overline{P}$$

Theorem 25

if P is a partial function, the above theorem can be strengthened to an equation

$$P \backslash R = R;(\overline{P}\backslash\overline{I}) \cup (P \backslash 0)$$

Proof: $RHS;P = R;(\overline{P}\backslash\overline{I});P \cup (P \backslash 0);P$ the distributive law of ;

$\subseteq R;(P;\overline{I})\backslash\overline{I};P \cup (P \backslash 0);P$ theorem 7

$= R;(P \backslash (\overline{I}\backslash\overline{I}));P \cup 0$ theorem 2

$= R;(P \backslash I);P$ theorem 9

$\subseteq R$ axiom 1

$\therefore RHS \subseteq P \backslash R$

Corollary

If P is a partial function, then

$$P^{+} \backslash R = R;(\overline{P}\backslash\overline{I}) \cup ((P \backslash 0) \cap (U \backslash R))$$

Proof: LHS $= (P \backslash R) \cap (\overline{P \backslash 0} \cup U \backslash R)$ theorem 22

$= (R ; (\bar{P} \backslash \bar{I}) \cup P \backslash 0) \cap (\overline{P \backslash 0} \cup U \backslash R)$ theorem 25

$= (R ; (\bar{P} \backslash \bar{I}) \cap \overline{P \backslash 0}) \cup (R ; (\bar{P} \backslash \bar{I}) \cap U \backslash R)$

$\cup ((P \backslash 0) \cap (U \backslash R))$ set theory

$= (R ; (\bar{P} \backslash \bar{I}) \cap U ; (\bar{P} \backslash \bar{I})) \cup (R ; (\bar{P} \backslash \bar{I}) \cap (U \backslash R))$

$\cup ((P \backslash 0) \cap (U \backslash R))$ corollary of theorem 5

$= R ; (\bar{P} \backslash \bar{I}) \cup ((P \backslash 0) \cap (U \backslash R))$ by the monotonicity of ;