DIJKSTRA'S PREDICATE TRANSFORMERS AND SMYTH'S POWERDOMAINS

G.D. Plotkin

Dept. of Computer Science
University of Edinburgh
Edinburgh EH9 3JZ

1. Introduction

In his book, "A Discipline of Programming" Dijkstra introduces a simple non-deterministic language based on his idea of guarded commands. He then introduces the idea of weakest preconditions on states to give his language a semantics called the predicate transformer semantics. This is very well-suited to showing that systems can achieve certain goals, that is to questions of correctness, and the whole thing fits well within the paradigm of denotational semantics where we can regard predicates and conditions as a kind of continuation. (See [Stra], [Gor], [Mil1], [Sto] for an account of continuations and denotational semantics, generally, and [de B], [de R], [Jen] for predicate transformer semantics in this style.)

On the other hand one can give the semantics of simple imperative languages using state transformation functions, and employing complete partial orders with a least element to handle divergence (nontermination). This kind of semantics can be considered as a direct abstraction from an operational semantics given, say, via an abstract machine. Within this framework, nondeterministic state transformations can be handled using powerdomains ([Plo2], [Smy]) which are a weak analogue of powersets for the complete partial orders under consideration.

The relation between these approaches was considered by de Roever who showed, in [de R], how, given a nondeterministic state-transformation, m, and a predicate, R, one could define the weakest-precondition, wp(m,R) of R, relative to m; this work employed Plotkin's powerdomain, based on the Egli-Milner order. He showed too that wp(m,R) was even continuous in m and R, and indicated that this enabled one to obtain the predicate transformer semantics (for a variant of Dijkstra's language including recursion) from the state transformation one. This was all fully worked out (among other things) by de Bakker in [de B]. Wand and Back have made other relevant and important contributions ([Wan], [Bac1]) and these will be discussed later.

In the present paper we regard this work as showing a homomorphism from the state transformation view to the predicate transformer one. By refining the definitions a little we succeed in strengthening the homomorphism to an isomorphism (for a slight variant of Dijkstra'a language). This involves, on the one hand, using the properties of predicate transformers, given by Dijkstra in chapter 3 of his book, to define the partial order of predicate transformers and, on the other hand, replacing the Egli-Milner order by the Smyth powerdomain.

However Dijkstra did not consider any particular language when introducing predicate transformers. Rather he used a general, somewhat informal, idea of mechanism. He gave some discussion of the relation between this view and a direct definition for the case of his language of guarded commands. We complete this discussion by showing how programs in our variant of his language can be regarded as mechanisms and so obtain another definition of their weakest precondition semantics. By examining the relationship between programs considered as mechanisms and their state transformation semantics we are able to demonstrate that both definitions of the predicate transformer semantics come to the same thing, thereby, in a sense, justifying Dijkstra's ideas.

In section 2 of this paper we present a formalisation of Dijkstra's mechanisms using the well-known idea of a transition relation. In section 3 we introduce our variant of Dijkstra's language of guarded commands and show how programs can act as mechanisms by defining the appropriate transition relations; this is the operational semantics of the language. In section 4 we introduce the partial order, PT, of predicate transformers and use it to give the predicate transformer semantics of our language. In section 5 we use Smyth's powerdomains to define the partial order, ST, of state transformations and use that to give the other semantics for our language; then we show that PT is isomorphic to ST (theorem 5.9) and this is even an isomorphism of the two semantics (theorem 5.12).

In section 6 we show how the operational semantics relates to the state transformation semantics (theorem 6.1) and use this result to demonstrate that the direct definition of the predicate transformer semantics gives the same result as the definition via the operational semantics (corollary 6.2).

Acknowledgements.

I thank Robin Milner and Willem de Roever for helpful comments and discussions, and Dines Björner and Cliff Jones for encouraging me to write this paper.

2. Transition Relations

In the chapter entitled, "The Characterisation of Semantics" of his book, Dijkstra discusses mechanisms, S, and their semantics by considering the weakest precondition on states, wp(S,R), of a post-condition R on states relative to a mechanism, S. He defines this by saying, for example:

"The condition that characterizes the set of all initial states such that activation will certainly result in a properly terminating happening leaving the system in a final state satisfying a given post-condition is called "the weakest pre-condition corresponding to that post-condition"."

This is admirably clear and perfectly precise once we know what conditions and mechanisms are.

For conditions we first postulate a denumerable set, States, of states, ranged over by the variable s; for the present purposes it is not necessary to have any particular structure on the set of states. Now we can choose between an intensional or an extensional view of conditions (= predicates). The emphasis on the extensional view at the end of chapter 2 of [Dij] leads us to take the set, Pred, of predicates as:

$$\text{Pred} = \mathcal{P}(\text{States})$$

the collection of all sets of states (ranged over by P, Q and R). Note that we have taken all subsets of States as there seems to be no reason here to exclude any particular ones. We shall often regard Pred as a complete lattice under the subset ordering, $\subseteq$. For work on the intensional point of view, see [Bac2,Mil, de B].

It remains to discuss mechanisms, S. From the above description of wp(S,R) all we need to know about S is, for any given initial state s, whether or not it will certainly result in a properly terminating happening and, if so, whether every possible final state satisfies R. To take exactly this view would lead us to considering S as a state-transformation function of some kind; this is the idea of de Roever and we take it up again in a later chapter. However it expresses little of the idea of what mechanisms and their happenings are. We intend to make that idea precise by considering mechanisms as performing, somehow, transitions from one state to another. Our method is a variant on a well-known game (e.g. [Kel]): we postulate a set, Systems, of systems (= machines = mechanisms), ranged over by S, and then define the set of configurations, ranged over by C, to be:

$$\text{Conf} = (\text{Systems} \times \text{States}) \cup \text{States}$$

Next we postulate a transition relation, $\rightarrow$: Conf $\times$ Conf and assume that no relationship of the form $s \rightarrow C$ holds. The idea is that configurations, C, represent states of a computation (happening); those of the form, $\langle S,s \rangle$, are possible initial or intermediate states and those of the form, s, are final ones.

Further relationships, $C \to C'$, represent one step of a computation: those of the form $\langle S,s\rangle \to \langle S',s'\rangle$ are intermediate steps and those of the form $\langle S,s\rangle \to s$ are final steps.

Definition 2.1 A system, S, is blocked in state, s, iff there is no configuration C such that $\langle S,s\rangle \to C$.

A system S may not terminate from state s iff either there are S',s' such that $\langle S,s\rangle \to^* \langle S',s'\rangle$ and S' is blocked in state s' or else there is an infinite sequence of the form, $\langle S,s\rangle = C_0 \to \dots \to C_m \to \dots$.

Now, clearly, to say that a system S, for a given initial state s, will certainly result in a properly terminating happening just means that S must terminate from s.

The weakest precondition, wp(S,R), of S and R can now be formally defined as:

$$\mathrm{wp}(S,R) = \{s \in \mathrm{States} \mid S \text{ must terminate from } s \text{ and for any } s' \text{ in States if } \langle S,s\rangle \to^* s' \text{ then } s' \text{ is in } R\}$$

The corresponding notion of precondition is defined for P,S,R by

$$P[S]R \text{ iff } \forall s \in P.\ (S \text{ must terminate from } s \text{ and for any } s' \text{ in States if } \langle S,s\rangle \to^* s' \text{ then } s' \text{ is in } R)$$

Evidently P[S]R holds iff $P \subseteq \mathrm{wp}(S,R)$ does, justifying the terminology of weakest preconditions.

This is the place to recapitulate the basic properties of weakest preconditions which we state for any system S and predicates Q,R:

Property 1 Strictness $\mathrm{wp}(S,\emptyset) = \emptyset$

Property 2 Monotonicity If $Q \subseteq R$ then $\mathrm{wp}(S,Q) \subseteq \mathrm{wp}(S,R)$

Property 3 Multiplicativity $\mathrm{wp}(S,Q) \cap \mathrm{wp}(S,R) = \mathrm{wp}(S,Q \cap R)$

The arguments for these properties are just (formal versions of) those already given by Dijkstra; note that multiplicativity implies monotonicity. For the last property we postulate further that the transition relation is finitary; this means that for any configuration C the set $\{C' \mid C \to C'\}$ is finite. The point of this condition is that it is then an easy consequence of König's lemma that for any S,s if S must terminate from s then the set $F(S,s) =_{\mathrm{def}} \{s' \mid \langle S,s\rangle \to^* s'\}$ is finite and nonempty.

Property 4 Continuity Let S be in Systems and let $Q_0 \subseteq Q_1 \subseteq \dots$ be an increasing sequence of predicates. Then,

$$\mathrm{wp}(S, \bigcup Q_i) = \bigcup \mathrm{wp}(S,Q_i)$$

This important property is considered in the chapter entitled, "On Nondeterminacy Being Bounded" where it is established for the guarded command language and a direct definition of its weakest precondition semantics. To establish the property in the present context note first that it follows by monotonicity that

$wp(S, \bigcup Q_i) \supseteq \bigcup wp(S,Q_i)$. For the converse suppose s is in $wp(S, \bigcup Q_i)$; then S must terminate from s and $F(S,s) \subseteq \bigcup Q_i$. But by the above remarks $F(S,s)$ is finite and so for some i we have $F(S,s) \subseteq Q_i$ showing that s is in $wp(S,Q_i)$ and establishing the converse. By the way, it should be clear that these properties should not be taken as an attempt to characterise or axiomatise wp(S,R) in any way; for example if we just set $wp'(S,R) = \emptyset$ that too would possess all the required properties.

With respect to [Har] our position is that, in his terminology, we are interested only in depth-first execution without backtracking. Consequently failures are as bad a way not to terminate as any other and we have not distinguished them, even at the present abstract level; they can be considered as being lumped in with the blockings. However, with this view Harel's language is, as he points out, too weak even to write conditionals and we prefer Dijkstra's. Note that we will find it useful to consider failure when discussing <u>guarded</u> commands.

Finally we consider preconditions and show one sense in which predicate transformer semantics can be considered as abstracting from correctness considerations. Now if P[S]R holds this can be regarded as a positive fact about S, or, following [Har], as a kind of lower bound on its behaviour, in that it says that certain things <u>will</u> happen. So it is natural to define a quasiorder (transitive and reflexive relation) on systems by putting for any S,S':

$$S \underset{\sim}{\sqsubseteq} S' \text{ iff } \forall P, R \in \text{Pred}.\ P[S]R \Rightarrow P[S']R$$

Clearly,

$$\begin{aligned} S \underset{\sim}{\sqsubseteq} S' &\equiv \forall R \in \text{Pred}.wp(S,R) \subseteq wp(S',R) \\ &\equiv \forall R \in \text{Pred}.R \text{ finite} \Rightarrow (wp(S,R) \subseteq wp(S',R)) \\ &\equiv \forall P,R \in \text{Pred}.P,R \text{ finite} \Rightarrow (P[S]R \Rightarrow P[S']R) \end{aligned}$$

(the last two equivalences use property 4). This justifies our using the subset ordering on predicates and the last two equivalences show there is no real harm in considering all predicates since it comes to the same thing if we restrict ourselves to the finite ones.

The quasiorder, $\underset{\sim}{\sqsubseteq}$, was also studied in [Bac1] as the <u>total correct refinement</u> relation, ref_T (more exactly, Back considered ref_T as a relation between certain kinds of state transformation functions and identified programs with such functions via their behaviour as systems). This work considered $\underset{\sim}{\sqsubseteq}$ as one of a family of refinement relations between programs and is part of a formalisation of the stepwise refinement technique as developed by Dijkstra and Wirth.

Note that we can define an equivalence, $\cong$, over systems by: $S \cong S'$ iff $S \underset{\sim}{\sqsubseteq} S' \underset{\sim}{\sqsubseteq} S$ and then:

$$\begin{aligned} S \cong S' &\equiv \forall P,R \in \text{Pred}.P[S]R \Leftrightarrow P[S']R \\ &\equiv \forall P,R \in \text{Pred}.P,Q \text{ finite} \Rightarrow (P[S]R \Leftrightarrow P[S']R) \end{aligned}$$

$$\equiv \forall R \in Pred. wp(S,R) = wp(S',R)$$
$$\equiv \forall R \in Pred. R \text{ finite} \Rightarrow (wp(S,R) = wp(S',R))$$

Thus if two systems are the same as predicate transformers then they satisfy the same correctness properties and vice versa. Thus predicate transformer semantics abstracts exactly from the desired correctness properties. Of course, changing the correctness properties considered might very well change the semantics needed and indeed the idea is very general and has, apart from the present case, not been much investigated.

3. Operational Semantics

Now we apply the ideas of the last section to a variant of the simple nondeterministic language of guarded commands presented by Dijkstra. The syntax of this language is presented in terms of four sets:

1. ACom is the set of atomic (= primitive) commands, assumed given; it is ranged over by the variable A.

2. BExp is the set of Boolean expressions, assumed given; it is ranged over by the variable B.

3a) Stmts is the set of statements; it is ranged over by the variable S.
 b) GCom is the set of guarded commands; it is ranged over by the variable G.

The sets, Stmts and GCom, are given together by the grammar:

$$S ::= A \mid \underline{\text{skip}} \mid \underline{\text{abort}} \mid (S;S) \mid \underline{\text{if}}\ G\ \underline{\text{fi}} \mid \underline{\text{do}}\ G\ \underline{\text{od}}$$
$$G ::= \underline{\text{empty}} \mid B \rightarrow S \mid (G \mathbin{[\!]} G)$$

The sets ACom and BExp have been left unspecified; different languages would be obtained via different choices. Typical elements could be assignments to variables such as "x:=x+y" or "x,y:=y,x" or to array elements such as "ar:(j)=ar(j)-1" in the case of ACom or simple conditions such as "$x \geq y$" or compound ones such as "x=5 $\underline{\text{and}}$ $ar(j) \leq u$" in the case of BExp. So an example statement could be:

$$\underline{\text{do}}\ (x>y \rightarrow x:=x-y \mathbin{[\!]} y>x \rightarrow y:=y-x)\ \underline{\text{od}}$$

This is all much as given by Dijkstra except that we do not consider block structure. There are also some minor differences in the syntax and in particular we painfully build up sequences of statements and guarded command sets by binary means; on the other hand we will often omit the brackets when it makes no difference to our assertions how they are understood. Note too that we explicitly allow compound guarded commands as this will make good semantic sense. This is perhaps the place to remark that guarded commands have a little more prominence in our treatment than in Dijkstra's, but nonetheless they still have rather a secondary role compared with statements themselves.

The operational semantics is specified by leaving States unanalysed and taking Systems to be Stmts and defining a transition relation. To do this we assume we are given two functions

$$\mathcal{A} : \text{ACom} \rightarrow (\text{States} \rightarrow \text{States})$$
$$\mathcal{B} : \text{BExp} \rightarrow (\text{States} \rightarrow T)$$

where T is the set, $\{tt, ff\}$, of truthvalues; these provide, in a rather abstract way, the semantics of atomic commands and Boolean expressions. Clearly $\mathcal{A}[\![A]\!](s) = s'$ means that the atomic command A changes state s to state s' and, similarly, $\mathcal{B}[\![B]\!](s) = t$ means that the Boolean expression B has value t in state s; note the typical use of emphatic brackets where the arguments of functions are syntactic objects. The totality of all the functions $\mathcal{A}[\![A]\!]$ and $\mathcal{B}[\![B]\!]$ means that we are assuming atomic commands always terminate properly when activated and the evaluation of Boolean expressions also always terminates properly. On the other hand, we could easily adjust the present theory to allow the possibility of non-termination.

Now we define the transition relation together with an auxiliary relation,

$$\rightarrow : (\mathrm{GCom} \times \mathrm{States}) \times ((\mathrm{Stmts} \times \mathrm{States}) \cup \{\underline{\mathrm{fail}}\})$$

with the same name. This is achieved by giving a little formal system of axioms and rules to show what relationships hold.

I1. $\langle A,s\rangle \rightarrow \mathcal{A}[\![A]\!](s)$

II1. $\langle \underline{\mathrm{skip}},s\rangle \rightarrow s$

III1. $\dfrac{\langle S_1,s\rangle \rightarrow \langle S_1',s'\rangle}{\langle (S_1;S_2),s\rangle \rightarrow \langle (S_1';S_2),s'\rangle}$ 2. $\dfrac{\langle S_1,s\rangle \rightarrow s'}{\langle (S_1;S_2),s\rangle \rightarrow \langle S_2,s'\rangle}$

IV1. $\dfrac{\langle G,s\rangle \rightarrow \langle S,s\rangle}{\langle \underline{\mathrm{if}}\ G\ \underline{\mathrm{fi}},s\rangle \rightarrow \langle S,s\rangle}$

V1. $\dfrac{\langle G,s\rangle \rightarrow \langle S,s\rangle}{\langle \underline{\mathrm{do}}\ G\ \underline{\mathrm{od}},s\rangle \rightarrow \langle (S;\ \underline{\mathrm{do}}\ G\ \underline{\mathrm{od}}),s\rangle}$ 2. $\dfrac{\langle G,s\rangle \rightarrow \underline{\mathrm{fail}}}{\langle \underline{\mathrm{do}}\ G\ \underline{\mathrm{od}},s\rangle \rightarrow s}$

VI1. $\langle \underline{\mathrm{empty}},s\rangle \rightarrow \underline{\mathrm{fail}}$

VII1. $\langle B \rightarrow S,s\rangle \rightarrow \langle S,s\rangle$ (if $\mathcal{B}[\![B]\!](s) = tt$)

2. $\langle B \rightarrow S,s\rangle \rightarrow \underline{\mathrm{fail}}$ (if $\mathcal{B}[\![B]\!](s) = ff$)

VIII1. $\dfrac{\langle G_i,s\rangle \rightarrow \langle S,s\rangle}{\langle (G_1 \Box G_2),s\rangle \rightarrow \langle S,s\rangle}$ (for $i=1,2$)

2. $\dfrac{\langle G_1,s\rangle \rightarrow \underline{\mathrm{fail}},\ \langle G_2,s\rangle \rightarrow \underline{\mathrm{fail}}}{\langle (G_1 \Box G_2),s\rangle \rightarrow \underline{\mathrm{fail}}}$

Hopefully these rules are in accord with the reader's intuitions: atomic commands do as they are assumed to do; skip does nothing; abort is always blocked as it has no rules or axioms; to execute the composition of two commands, execute the first and then the second. If the remaining rules are not so immediate consider instead the following derived rules:

IX1. $\langle \underline{\mathrm{if}}\ B_1 \rightarrow S_1 \Box \ldots \Box\ B_n \rightarrow S_n\ \underline{\mathrm{fi}},s\rangle \rightarrow \langle S_i,s\rangle$
(if $\mathcal{B}[\![B_i]\!](s) = tt$ where $1 \le i \le n$)

X1. $\langle \underline{\mathrm{do}}\ \underline{\mathrm{empty}}\ \underline{\mathrm{od}},s\rangle \rightarrow s$

2. $\langle \underline{\mathrm{do}}\ B_1 \rightarrow S_1 \Box \ldots \Box\ B_n \rightarrow S_n\ \underline{\mathrm{od}},s\rangle \rightarrow \langle S_i;\ \underline{\mathrm{do}}\ B_1 \rightarrow S_1 \Box \ldots \Box\ B_n \rightarrow S_n\ \underline{\mathrm{od}},s\rangle$
(if $\mathcal{B}[\![B_i]\!](s) = tt$ where $1 \le i \le n$)

3. $\langle \underline{\mathrm{do}}\ B_1 \rightarrow S_1 \Box \ldots \Box\ B_n \rightarrow S_n\ \underline{\mathrm{od}},s\rangle \rightarrow s$
(if $\mathcal{B}[\![B_i]\!](s) = ff$ whenever $1 \le i \le n$)

These "non-binary" rules can replace IV - VIII as far as statements are concerned. Note that if G fi blocks when the guarded command G fails, whereas do G od terminates, as expected; note too that nondeterminism is introduced in rule VIII1 (or, alternatively, IX1 and X2).

As an example let G_1 be $x>y \rightarrow x:=x-y$ and let G_2 be $y>x \rightarrow y:=y-x$ and take G to be

G_1 [] G_2; let States, $\mathcal{R}$ and $\mathcal{S}$ have the (hopefully) evident definitions. Now, if m>n then

<do G od,<m,n>> -> <x:=x-y; do G od,<m,n>> (by X2)
-> <do G od,<m-n,n>> (by III2 ,I1)

and if m=n then

<do G od,<m,n>> -> <m,n> (by X3)

and if m<n then

<do G od,<m,n>> -> <m,n-m> (by X2 , III2 , I1)

and these are the only possibilities. For other examples of this method of giving operational semantics see [Plo1],[Mil3],[Hen].

As an example of equivalent statements consider:

$S_1 =_{def}$ do x=x -> y:=y od
$S_2 =_{def}$ x:=1; do x>0 -> x:=0 [] x>0 -> x:=1 od
$S_3 =_{def}$ x,y:=1,0; do y=0 -> y:=1 [] y=0 -> x:=x+1 od

The point here is that for any initial state any of these statements may not terminate and so we always have $wp(S_i,R) = \emptyset$. Therefore abort $\cong S_1 \cong S_2 \cong S_3$.

As an example of the quasiorder, $\underset{\sim}{\sqsubset}$, consider:

S_4 = if x=x -> x:=0 [] x=x -> x:=1 fi
S_5 = x:=0

Here $S_4 \underset{\sim}{\sqsubset} S_5$ since producing more final states (for a given initial one) means that fewer preconditions hold. This somewhat strange reversal of orders is also characteristic of the Smyth powerdomain and explains why it appears in the present setting.

4. Predicate Transformers

Given any mechanism S we can "fix" the first argument of wp to obtain a predicate transformer, f_S: Pred -> Pred where:

$$f_S(R) = wp(S,R)$$

This section gives a direct definition of the predicate transformers for our language following [Dij § 4]. Of course we could always turn this around, saying that this section specifies the predicate transformer semantics and any implementation - in particular the abstract one in section 3 - must be in accordance with it (see [Dij § 26]). In any case the relation between the two definitions is clearly what is important.

Following a standard denotational approach we first decide what the collection, PT, of all predicate transformers is to be. It will be useful to first recollect a few standard ideas (see [Wad] for a leisurely account).

We will make some use of the typed λ-calculus so that if x ranges over the set X and .....x.... is an expression denoting an element of the set Y (possibly involving x) then the expression $\lambda x \in X.$ ....x.... (or equivalently, $\lambda x.$ ....x...., if X can be understood from the context) denotes the function f: X -> Y where, for any x in X:

$$f(x) = \ldots.x\ldots. \,.$$

We also need a few definitions concerning partial orders.

Definition 4.1 1. A complete partial order (cpo) is a partial order, $\langle D, \sqsubseteq\rangle$ with a least element, $\perp_D$, in which every increasing sequence $d_0 \sqsubseteq d_1 \sqsubseteq \ldots$ has a least upper bound, $\bigsqcup_D d_i$; the greatest lower bound of two elements d and d' is, if it exists, written as $d \sqcap d'$.

2. Let f: D -> E be a function from one cpo D to another, E. Then f is strict if $f(\perp_D) = \perp_E$; it is monotonic iff whenever d and d' are elements of D with $d \sqsubseteq d'$ then $f(d) \sqsubseteq f(d')$; it is continuous if it is monotonic and for every increasing chain $d_0 \sqsubseteq d_1 \sqsubseteq \ldots$ of elements of D we have: $f(\bigsqcup_D d_i) = \bigsqcup_E f(d_i)$; it is multiplicative if whenever d and d' are elements of D such that $d \sqcap d'$ exists then $f(d) \sqcap f(d')$ exists and $f(d \sqcap d') = f(d) \sqcap f(d')$.

For example Pred is a cpo with $\perp = \emptyset$, $\bigsqcup d_i = \bigcup d_i$ and $d \sqcap d' = d \cap d'$. It is well-known that the identity id_D: D -> D (where id(d) = d) is strict, continuous and multiplicative as is $\perp_{D,E}$: D -> E (where $\perp_{D,E}(d) = \perp_E$); further if f: D -> E and g: E -> F are strict (monotonic, continuous, multiplicative) so is their composition $g \circ f$: D -> F. Finally, if f: D -> D is continuous it has a least fixed-point, namely $Y(f) = \bigsqcup_D f^k(\perp_D)$.

Rather than take PT to be the collection of continuous functions from Pred to Pred as do some authors ([de B],[Jen]) we follow the properties of weakest preconditions discussed in section 2 and put:

$PT = \{f\colon Pred \to Pred \mid f \text{ is strict, continuous and multiplicative}\}$

and turn it into a partial order by the pointwise ordering:

$f \sqsubseteq g$ iff $\forall R \in Pred.\ f(R) \subseteq g(R)$.

<u>Proposition 4.2</u> 1. The partial order, PT, is a cpo with least element $\perp_{Pred,Pred}$ and with least upper bounds of increasing chains $f_0 \sqsubseteq f_1 \sqsubseteq \ldots$ given by: $(\bigsqcup f_i)(R) = \bigsqcup f_i(R)$. Further greatest lower bounds of pairs f, g of elements always exist, being given by: $(f \sqcap g)(R) = f(R) \cap g(R)$.

2. The identity, id_{Pred} is a predicate transformer and so is the composition of any two predicate transformers.

3. Let f,g be in PT and let P and Q be disjoint predicates. Then h is a predicate transformer, where $h(R) = (P \cap f(R)) \cup (Q \cap g(R))$.

<u>Proof</u> 1. We have already observed that $\perp_{Pred,Pred}$ is a predicate transformer and it clearly must then be the least one. For any increasing sequence, $f_0 \sqsubseteq f_1 \sqsubseteq \ldots$ of predicate transformers put $f(R) = \bigcup f_i(R)$. It is well-known that this defines a strict continuous function as the f_i are all strict and continuous. For multiplicativity we calculate for any Q,R:

$$\begin{aligned} f(Q \cap R) &= \bigcup_i f_i(Q \cap R) = \bigcup_i f_i(Q) \cap f_i(R) \\ &= \bigcup f_i(Q) \cap \bigcup f_i(R) \quad (\text{as } f_0(Q) \subseteq f_1(Q) \subseteq \ldots) \\ &= f(Q) \cap f(R). \end{aligned}$$

2. This has already been observed.

3. It is clear that h is strict and continuity and multiplicativity are easy calculations. For example for the latter:

$$\begin{aligned} h(R \cap R') &= (P \cap f(R) \cap f(R')) \cup (Q \cap g(R) \cap g(R')) \\ &= (P \cap f(R) \cap P \cap f(R')) \cup (P \cap f(R) \cap Q \cap g(R')) \cup \\ &\quad (Q \cap g(R) \cap P \cap f(R')) \cup (Q \cap g(R) \cap Q \cap g(R')) \\ &= [(P \cap f(R)) \cup (Q \cap g(R))] \cap [(P \cap f(R')) \cup (Q \cap g(R'))] \\ &= h(R) \cap h(R'). \ \blacksquare \end{aligned}$$

Now we define a few operators in order to give the semantics in a more or less algebraic way, following [ADJ]. First define two sets, d-ST, of <u>deterministic state-transformation functions</u> and Bool by:

d-ST = States -> States

Bool = States -> T

and for any p,q in Bool put $p^+ = p^{-1}(tt)$, $p^- = p^{-1}(ff)$ and define $p \vee q$ by:

$(p \vee q)(s) = p(s) \vee q(s)$

where we are using logical disjunction on the right. It is worth remarking here that we need Bool as well as Pred since the ordering on Pred is not appropriate for values of Boolean expressions: if we wanted non-terminating Boolean expressions we would rather use the partial order, States -> $T_\perp$ (see the next section for definitions).

<u>Conversion</u> Define Conv: d-St -> PT by:

$$\mathrm{Conv}(m)(R) = m^{-1}(R)$$

and it is an easy calculation that Conv(m), so defined, is in PT.

<u>Composition</u> Define Comp: PT^2 -> PT by:

$$\mathrm{Comp}(f,g) = f \bullet g$$

and Proposition 4.2.2 assures us that Comp(f,g) is in PT.

<u>Conditional</u> Define Cond: Bool $\times$ PT -> PT by:

$$\mathrm{Cond}(p,f)(R) = p^+ \cap f(R)$$

and since $\mathrm{Cond}(p,f)(R) = (p^+ \cap f(R)) \cup (p^- \cap \perp_{PT}(R))$

Proposition 4.2.3 assures us it is in PT.

<u>Iteration</u> Define Do: Bool $\times$ PT -> PT by:

$$\mathrm{Do}(p,f)(R) = Y(\lambda Q \in \mathrm{Pred}.(p^{-1} \cap R) \cup (p^+ \cap f(Q)))$$

What this means is that for p,f and any R if we define the function h_R on predicates by:

$$h_R(Q) = (p^- \cap R) \cup (p^+ \cap f(Q))$$

we observe that h_R is continuous and so has a least fixed-point $Y(h_R) = \bigcup h_R^k(\emptyset)$, which is what we take the value of Do(p,f) at R to be.

So if we define $\mathrm{Do}_k(p,f)$: Pred -> Pred by

$$\mathrm{Do}_k(p,f)(R) = h_R^k(\emptyset)$$

then $\mathrm{Do}(p,f)(R) = \bigcup \mathrm{Do}_k(p,f)(R)$. Now by induction on k one easily sees that each $\mathrm{Do}_k(p,f)$ is in PT. This is clear for k=0 and for k+1 we observe:

$$\begin{aligned}\mathrm{Do}_{k+1}(p,f)(R) &= h_R(\mathrm{Do}_k(p,f)(R)) \\ &= (p^- \cap R) \cup (p^+ \cap f(\mathrm{Do}_k(p,f)(R)))\end{aligned}$$

and so, by the induction hypothesis and Proposition 4.2.3, $\mathrm{Do}_{k+1}(p,f)$ is also in PT. It now follows by Proposition 4.2.1 that Do(p,f) is also a predicate transformer and so is well-defined.

<u>Bar</u> We introduce a function which will give semantic significance to Dijkstra's bar symbol, $[\!]$. Define Bar: $(\mathrm{Bool} \times \mathrm{PT})^2$ -> (Bool $\times$ PT) by:

$$\begin{aligned}&\mathrm{Bar}(\langle p,f\rangle,\langle q,g\rangle) = \\ &\quad \langle p \vee q, \lambda R \in \mathrm{Pred}.(p^+ \cap [(q^+ \cap f(R) \cap g(R)) \cup (q^- \cap f(R))]) \\ &\qquad \cup (p^- \cap q^+ \cap g(R)\rangle\end{aligned}$$

The arrangement of this definition and Proposition 4 shows that the second component is indeed a predicate transformer; it can also be written as:

$\lambda R \in \text{Pred}.\ (p^+ \cup q^+) \cap (p^- \cup f(R)) \cap (q^- \cup g(R))$

Thus Bar is clearly commutative; one can also show that it is associative and further, using an obvious notation that, for n>0:

$$\text{Bar}\langle p_i, f_i\rangle_{i=1,n} = \langle \bigvee_{i=1,n} p_i, \lambda R \in \text{Pred}(\bigcup_{i=1,n} p_i^+) \cap \bigcap_{i=1,n} (p_i^- \cup f_i(R))\rangle$$

On the other hand, Bar is not absorptive; indeed as $\text{Bar}(\langle p,f\rangle, \langle p,f\rangle) = \langle p, \lambda R \in \text{Pred}.\ p^+ \cap f(R)\rangle$ absorption holds at $\langle p,f\rangle$ iff $f(R) \subseteq p^+$ for any R. Similarly the natural zero $\langle \lambda s \in \text{States}.\ ff, \perp_{PT}\rangle$ does not work as $\text{Bar}(\langle \lambda s.ff, \perp_{PT}\rangle, \langle p,f\rangle) = \langle p, \lambda R. p^+ \cap f(R)\rangle$ too.

It is now straightforward to present a classical denotational semantics for predicate transformers by defining two denotation functions.

$\mathcal{C}_{PT}$: Stmts -> PT

$\mathcal{G}_{PT}$: GCom -> Bool × PT

The idea for statements is that where Dijkstra would write

wp(S,R) = ... R ...

one writes instead

$\mathcal{C}_{PT}[\![S]\!] = \lambda R \in \text{Pred}.\ \ldots R \ldots$

and so $\mathcal{C}_{PT}$ is just a Curried version of wp. The idea for guarded commands is that if $\mathcal{G}_{PT}[\![G]\!] = \langle p,f\rangle$ then p is the meaning of an implicit guard of G and f is a predicate transformer for an implicit command of G.

The denotation functions are defined by a mutual structural induction on statements and guarded commands via the following clauses:

C1 $\mathcal{C}_{PT}[\![A]\!] = \text{Conv}(\mathcal{A}[\![A]\!])$

C2 $\mathcal{C}_{PT}[\![\underline{\text{skip}}]\!] = \text{id}_{PT}$

C3 $\mathcal{C}_{PT}[\![\underline{\text{abort}}]\!] = \perp_{PT}$

C4 $\mathcal{C}_{PT}[\![(S_1;S_2)]\!] = \text{Comp}(\mathcal{C}_{PT}[\![S_1]\!], \mathcal{C}_{PT}[\![S_2]\!])$

C5 $\mathcal{C}_{PT}[\![\underline{\text{if}}\ G\ \underline{\text{fi}}]\!] = \text{Cond}(\mathcal{G}_{PT}[\![G]\!])$

C6 $\mathcal{C}_{PT}[\![\underline{\text{do}}\ G\ \underline{\text{od}}]\!] = \text{Do}(\mathcal{G}_{PT}[\![G]\!])$

G1 $\mathcal{G}_{PT}[\![\underline{\text{empty}}]\!] = \langle \lambda s \in \text{States}.\ ff, \perp_{PT}\rangle$

G2 $\mathcal{G}_{PT}[\![B \to S]\!] = \langle \mathcal{B}[\![B]\!], \mathcal{C}_{PT}[\![S]\!]\rangle$

G3 $\mathcal{G}_{PT}[\![(S_1 \,[\!]\, S_2)]\!] = \text{Bar}(\mathcal{G}_{PT}[\![S_1]\!], \mathcal{G}_{PT}[\![S_2]\!])$

Just as in the case of the operational semantics it is not necessary to define a semantics for guarded commands if we do not mind a certain clumsiness of expression. For as is easily seen from the above remarks the following holds and could replace C5:

C7 $\mathcal{C}_{PT}[\![\underline{\text{if}}\ \underline{\text{empty}}\ \underline{\text{fi}}]\!] = \perp_{PT}$

C8 $\mathcal{C}_{PT}[\![\underline{if}\ B_1 \to S_1 \,\square\, \ldots . \,\square\, B_n \to S_n\ \underline{fi}]\!] =$
$\lambda R \in \text{Pred.}(\bigcup_{i=1,n} \mathcal{B}[\![B_i]\!]^+) \cap \bigcap_{i=1,n}(\mathcal{B}[\![B_i]\!]^- \cup \mathcal{C}_{PT}[\![S_i]\!](R))$

and for C6 we have:

C9 $\mathcal{C}_{PT}[\![\underline{do}\ \underline{empty}\ \underline{od}]\!] = id_{PT}$

C10 $\mathcal{C}_{PT}[\![\underline{do}\ B_1 \to S_1 \,\square\, \ldots . \,\square\, B_n \to S_n]\!] = \lambda R \in \text{Pred.}\bigcup_k D_k$
(where $D_0(R) = \emptyset$ and

$$D_{k+1}(R) = [(\bigcup_{i=1,n} \mathcal{B}[\![B_i]\!]^+) \cap \bigcap_{i=1,n}(\mathcal{B}[\![B_i]\!]^- \cup \mathcal{C}_{PT}[\![S_i]\!](D_k(R)))] \cup [(\bigcap_{i=1,n} \mathcal{B}[\![B_i]\!]^-) \cap R])$$

Dijkstra's H_{k+1} corresponds to our D_k .

Note the following equivalences which either follow from the above remarks or are easily proved directly:

$$\mathcal{C}_{PT}[\![(S_1;(S_2;S_3))]\!] = \mathcal{C}_{PT}[\![((S_1;S_2);S_3)]\!]$$
$$\mathcal{C}_{PT}[\![(\underline{skip};S)]\!] = \mathcal{C}_{PT}[\![(S;\ \underline{skip})]\!] = \mathcal{C}_{PT}[\![S]\!]$$
$$\mathcal{C}_{PT}[\![(\underline{abort};S)]\!] = \mathcal{C}_{PT}[\![(S;\underline{abort})]\!] = \mathcal{C}_{PT}[\![\underline{abort}]\!]$$
$$\mathcal{C}_{PT}[\![\ \underline{if}\ \underline{empty}\ \underline{fi}]\!] = \mathcal{C}_{PT}[\![\ \underline{abort}]\!]$$
$$\mathcal{C}_{PT}[\![\ \underline{do}\ \underline{empty}\ \underline{od}]\!] = \mathcal{C}_{PT}[\![\ \underline{skip}]\!]$$
$$\mathcal{G}_{PT}[\![(G_1 \,\square\, (G_2 \,\square\, G_3))]\!] = \mathcal{G}_{PT}[\![((G_1 \,\square\, G_2) \,\square\, G_3)]\!]$$
$$\mathcal{G}_{PT}[\![(G_1 \,\square\, G_2)]\!] = \mathcal{G}_{PT}[\![(G_2 \,\square\, G_1)]\!]$$

Among other things these equivalences allow us to disregard brackets when considering the semantics of statements and guarded commands. On the other hand the following two equivalences fail:

$$\mathcal{G}_{PT}[\![(G_1 \,\square\, G_1)]\!] = \mathcal{G}_{PT}[\![G_1]\!]$$
$$\mathcal{G}_{PT}[\![(\underline{empty} \,\square\, G_1)]\!] = \mathcal{G}_{PT}[\![G_1]\!]$$

as, for example, one could take G_1 to be x = 0 -> x:=1. It would be interesting to see a semantics for guarded commands which does not suffer from these problems. To further emphasize the secondary status of guarded commands, in the present treatment, note that everything is all right if we consider statement contexts, ... G ... as we do always have:

$$\mathcal{C}_{PT}[\![\ldots(G_1 \,\square\, G_1)\ldots]\!] = \mathcal{C}_{PT}[\![\ \ldots\ G_1\ \ldots]\!]$$
$$\mathcal{C}_{PT}[\![\ldots(\underline{empty} \,\square\, G_1)\ldots]\!] = \mathcal{C}_{PT}[\![\ldots\ G_1\ \ldots]\!].$$

5. State Transformations

We now look for a state transformation semantics which reflects the operational semantics more directly than do predicate transformers. Indeed we want a cpo, ST, of state transformation functions, which is isomorphic to PT. Because of the possibilities of nontermination and nondeterminism, the simple set d-ST = States-> States will not do. To handle nontermination we introduce the so-called flat cpos.

Definition 5.1 Let X be a set. The flat cpo $X_\perp$ is $X \cup \{\perp\}$ ordered by: $x \sqsubseteq y$ iff $x = \perp$ or $x = y$.

Clearly $X_\perp$ is a cpo with least element $\perp$ (used for nontermination) and with lubs of any increasing sequences $x_0 \sqsubseteq x_1 \sqsubseteq ..$ as all such sequences are eventually constant, having at most two different elements (hence the term "flat").

If it were not for nondeterminism we could take the collection of state transformations to be States -> $\text{States}_\perp$; instead we use multi-valued functions m: States -> $\mathcal{P}_S(\text{States}_\perp)$ where $\mathcal{P}_S(\text{States}_\perp)$ is a cpo of subsets of $\text{States}_\perp$.

Definition 5.2 Let X be a set. The Smyth powerdomain $\mathcal{P}_S(X_\perp)$ of $X_\perp$ is the set $\{x \subseteq X_\perp \mid x = X_\perp \text{ or } (x \neq \emptyset \text{ and } x \subseteq X \text{ and } x \text{ is finite})\}$ partially ordered by: $x \sqsubseteq y$ iff $x \supseteq y$.

Before discussing the lattice-theoretic properties of $\mathcal{P}_S(X\ )$ we examine the motivations for what is, at first sight, a somewhat curious definition. Using this definition we can define ST in such a way that state transformations m can capture exactly enough of the operational meaning of programs to determine their associated weakest preconditions

Definition 5.3 1. The partial order, ST, of state transformations is the set States -> $\mathcal{P}_S(\text{States}_\perp)$ ordered (pointwise) by:

$$m \sqsubseteq m' \text{ iff } \forall s \in \text{States}.\ m(s) \sqsubseteq m(s')$$

2. The weakest precondition wp(m,R) of a state transformation, m, relative to a predicate, R, is given by:

$$wp(m,R) = \{s \in \text{States} \mid \perp \notin m(s) \text{ and } m(s) \subseteq R\}$$

Now the first point is that if m is to be the denotation of a statement S then we expect that m(s) is to say what the results of the possible computations of S starting from s are, and also that $\perp$ is to record nontermination so that $\perp \in m(s)$ iff S may not terminate from s. The König's lemma argument in section 2 shows that if $\perp \notin m(s)$ then m(s) is finite and nonempty. So far, this would lead us to taking those nonempty subsets which are finite or contain $\perp$ and then to the Egli-Milner ordering on them; this is the approach used in [de R] and discussed more generally in [Plo2]. However for the purposes of giving weakest preconditions, we note that if $\perp \in m(s)$ and $\perp \in m(s')$ then m(s) and m(s') are equivalent in that both $s \in wp(m,R)$ and $s' \in wp(m,R)$ are impossible for any R. Hence we have simply identified such sets with each other in the definition of $\mathcal{P}_S(\text{States}_\perp)$, equating them all, for convenience, with $States_\perp$. This explains our choice of the elements of the Smyth powerdomain.

The choice of ordering follows the examples in section 3. Clearly $States_\perp$ should be the least element. For the other elements suppose $m(s) \supseteq m(s')$ and neither contains $\perp$. Then, for any R, if $s \in wp(m,R)$ then $s' \in wp(m,R)$; thus $m(s')$ is better than $m(s)$ in that it makes s' satisfy more preconditions and so we put $m(s) \sqsubseteq m(s')$, explaining the curious order reversal in the definition. For a general account of the Smyth powerdomain, and other motivations for its choice, see [Smy].

<u>Proposition 5.4</u> 1. The partial order $\mathcal{P}_S(X_\perp)$ is a cpo: it has least element $X_\perp$ and every increasing chain $x_0 \sqsubseteq x_1 \sqsubseteq \ldots$ is eventually constant with least upper bound $\bigsqcup x_i = \bigcap x_i$. Further any two elements x and y have greatest lower bound $x \sqcap y = x \cup y$ and, considered as a function of two arguments, the greatest lower bound is continuous in each argument.

2. The partial order, ST, is a cpo: it has least element $\lambda s \in States.\ \perp_{\mathcal{P}_S(States_\perp)}$ and every increasing chain $m_0 \sqsubseteq m_1 \sqsubseteq \ldots$ has least upper bound m, where $m(s) = \bigsqcup m_i(s)$.

<u>Proof</u> 1. As $X_\perp$ includes any other element it is the least element. If $x_0 \sqsubseteq x_1 \sqsubseteq \ldots$ is increasing and non-constant, it is eventually a decreasing sequence of finite sets, with respect to $\subseteq$, and so is eventually constant. It follows at once that $\bigsqcup x_i = \bigcap x_i$. It is easy to see that if x and y are elements of $\mathcal{P}_S(X_\perp)$ then so is $x \cup y$ and it follows at once from the definition of $\sqsubseteq$ that $x \sqcap y = x \cup y$; as for continuity it is clear that $x \cup y$ is monotonic in x and y and since every increasing sequence is eventually constant it must also be continuous in x and y.

2. Immediate from the pointwise definition of the partial order on ST and the fact, proved in part 1, that $\mathcal{P}_S(States_\perp)$ is a cpo. $\boxtimes$

There are two other useful basic functions aside from $\cup$; we restrict ourselves to $\mathcal{P}_S(States_\perp)$ although these functions exist in general.

<u>Singleton</u> The function $\{\!|\cdot|\!\}$: States $\to \mathcal{P}_S(States_\perp)$ is defined by:

$$\{\!|d|\!\} = \begin{cases} \{d\} & (d \neq \perp) \\ States_\perp & (d = \perp) \end{cases}$$

As $\{\!|\cdot|\!\}$ is monotonic and $States_\perp$ is flat it is continuous; generally we omit the vertical bars when writing $\{\!|\cdot|\!\}$.

<u>Application</u> The binary function App: $(ST \times \mathcal{P}_S(States_\perp)) \to \mathcal{P}_S(States_\perp)$ is defined by:

$$\mathrm{App}(m,x) = \begin{cases} \bigcup\{m(s) \mid s \in x\} & (\text{if } x \neq \bot \text{ and for any } s \text{ in } x \; m(s) \neq \bot) \\ \bot & (\text{otherwise}) \end{cases}$$

<u>Lemma 5.5</u> The function App is continuous in each argument. It is strict and multiplicative in its second argument and for any s in States, $\mathrm{App}(m,\{s\}) = m(s)$.

<u>Proof</u> It is a straightforward calculation, which we leave to the reader, that App is monotonic in its second argument (and therefore continuous in its second argument as all increasing sequences in $\mathcal{P}_S(\mathrm{States}_\bot)$ are eventually constant) and that it is strict and multiplicative in its second argument and that $\mathrm{App}(m,\{s\}) = m(s)$.

To see that App is continuous in its first argument, we begin with monotonicity and assume $m \sqsubseteq m'$ and show $\mathrm{App}(m,x) \sqsubseteq \mathrm{App}(m',x)$ for any x. If $x = \bot$ this follows as App is strict in its second argument and otherwise x is finite and we proceed by induction on the size of x. If x is a singleton, $\{s\}$, then $\mathrm{App}(m,x) = m(s) \sqsubseteq m'(s) = \mathrm{App}(m',x)$. Otherwise $x = x_0 \cup x_1$ where x_0 and x_1 are strictly smaller than x and so we can calculate:

$$\begin{aligned} \mathrm{App}(m,x_0 \cup x_1) &= \mathrm{App}(m,x_0) \cup \mathrm{App}(m,x_1) \sqsubseteq \mathrm{App}(m',x_0) \cup \mathrm{App}(m',x_1) \\ &= \mathrm{App}(m,x_0 \cup x_1). \end{aligned}$$

For continuity one just proceeds in the same way to show that for any increasing chain $m_0 \sqsubseteq m_1 \sqsubseteq \ldots$ and any x we have: $\mathrm{App}(\bigsqcup m_i,x) = \bigsqcup \mathrm{App}(m_i,x)$. ⊠

Now we can give the state transformation semantics, starting with the definition of operators similar to those in section 4 and with the same names. First, however, note the useful conditional function, defined for every cpo D,
(<u>if</u> · <u>then</u> · <u>else</u> ·): $T \times D \times D \to D$ where

$$\underline{\text{if}}\ t\ \underline{\text{then}}\ d\ \underline{\text{else}}\ d' = \begin{cases} d & (t = tt) \\ d' & (t = ff) \end{cases}$$

It is easily seen to be continuous in its second and third arguments.

<u>Conversion</u> Define Conv: d-ST -> ST by:

$$\mathrm{Conv}(m)(s) = \{m(s)\}$$

<u>Composition</u> Define Comp: $ST^2 \to ST$ by:

$$\mathrm{Comp}(m,m')(s) = \mathrm{App}(m',m(s))$$

From Proposition 5.4 2. and Lemma 5.5 we see that Comp(m,m') is continuous in m and m'.

<u>Conditional</u> Define Cond: Bool × ST -> ST by:

$$\mathrm{Cond}(p,m)(s) = \underline{\text{if}}\ p(s)\ \underline{\text{then}}\ m(s)\ \underline{\text{else}}\ \bot$$

<u>Iteration</u> Define Do: Bool × ST -> ST by:

$$\mathrm{Do}(p,m) = Y(\lambda m' \in ST.\lambda s \in \mathrm{States}.\ \underline{\mathrm{if}}\ p(s)\ \underline{\mathrm{then}}\ \mathrm{Comp}(m,m')(s)\ \underline{\mathrm{else}}\ \{s\})$$

What this means is that for any p and m we can define a continuous function, h, of state transformations by:

$$h(m')(s) = \underline{\mathrm{if}}\ p(s)\ \underline{\mathrm{then}}\ \mathrm{Comp}(m,m')(s)\ \underline{\mathrm{else}}\ \{s\}$$

and Do(p,m) is the least fixed-point, $\bigsqcup h^k(\bot_{ST})$, of h.
So if we define $\mathrm{Do}_k(p,m)$: ST by:

$$\mathrm{Do}_k(p,m) = h^k(\bot_{ST})$$

then $\mathrm{Do}(p,m) = \bigsqcup \mathrm{Do}_k(p,m)$ and the Do_k have the recursive definition, $\mathrm{Do}_0 = \bot_{ST}$ and:

$$\mathrm{Do}_{k+1}(p,m)(s) = \underline{\mathrm{if}}\ p(s)\ \underline{\mathrm{then}}\ \mathrm{Comp}(m,\mathrm{Do}_k(p,m))(s)\ \underline{\mathrm{else}}\ \{s\}.$$

<u>Bar</u> Define Bar: $(\mathrm{Bool} \times ST)^2$ -> (Bool × ST) by:

$$\begin{aligned}&\mathrm{Bar}(\langle p,m\rangle,\langle q,m'\rangle) = \\ &\quad \langle p \vee q, \lambda s \in \mathrm{States}. \\ &\qquad \underline{\mathrm{if}}\ p(s)\ \underline{\mathrm{then}}\ (\underline{\mathrm{if}}\ q(s)\ \underline{\mathrm{then}}\ m(s) \cup m'(s)\ \underline{\mathrm{else}}\ m(s)) \\ &\qquad\qquad \underline{\mathrm{else}}\ (\underline{\mathrm{if}}\ q(s)\ \underline{\mathrm{then}}\ m'(s)\ \mathrm{else}\ \bot)\rangle\end{aligned}$$

Now we can define the denotational semantics as usual via two functions:

$\mathcal{C}_{ST}$: Stmts -> ST

$\mathcal{G}_{ST}$: GCom -> Bool × ST.

The semantic clauses C1, C4, C5, C6, G2 and G3 are just as before (except with $\mathcal{C}_{PT}$, $\mathcal{G}_{PT}$ replaced by $\mathcal{C}_{ST}$, $\mathcal{G}_{ST}$ of course) and the others are:

C2 $\mathcal{C}_{ST}[\![\underline{\mathrm{skip}}]\!] = \lambda s \in \mathrm{States}.\{s\}$

C3 $\mathcal{C}_{ST}[\![\underline{\mathrm{abort}}]\!] = \bot_{ST}$

G1 $\mathcal{G}_{ST}[\![\underline{\mathrm{empty}}]\!] = \langle \lambda s \in \mathrm{States}.\ ff, \bot_{ST}\rangle$

We are now in a position to show first that ST and PT are isomorphic and second that the two semantics based on state transformations and predicate transformers are also isomorphic. Definition 5.3 2. will enable us to go from ST to PT and for the converse direction the key lemma is:

<u>Lemma 5.6</u> <u>Stability</u> Let f: Pred -> Pred be any map of predicates. Then f is in PT if and only if whenever s ∈ f(States) there is a finite non-empty set, min(f,s), satisfying the following condition:

$$\forall R \in \mathrm{Pred}.\ s \in f(R) \equiv \min(f,s) \subseteq R.$$

<u>Proof</u> Suppose f is in PT and s ∈ f(States). Since States is denumerable there is an increasing sequence $x_0 \subseteq x_1 \subseteq \ldots$ of finite sets of states such that States $= \bigcup x_i$. Then we have:

$$s \in f(\mathrm{States}) = f(\bigcup x_i) = \bigcup f(x_i)$$

by the continuity of f. Hence there is a finite set, x, such that $s \in f(x)$. Let min(f,s) be such a finite set of the smallest possible cardinality; it is non-empty as f is strict and furthermore we claim it satisfies the condition. For on the one hand if $\min(f,s) \subseteq R$ then $s \in f(\min(f,s)) \subseteq f(R)$, by the monotonicity of f; on the other hand if $s \in f(R)$ then, by the multiplicativity of f, $s \in f(R) \cap f(\min(f,s)) = f(R \cap \min(f,s))$ and so by the choice of min(f,s) we must have $R \cap \min(f,s) = \min(f,s)$ and so $\min(f,s) \subseteq R$.

Conversely suppose f: PT -> PT satisfies the proposed criterion. To see it is continuous, suppose $R_0 \subseteq R_1 \subseteq \dots$ is an increasing sequence. Then:

$$\begin{aligned} s \in f(\bigcup R_i) &\equiv \min(f,s) \subseteq \bigcup R_i \quad \text{(by the criterion)} \\ &\equiv \min(f,s) \subseteq R_i, \text{ for some } i \quad \text{(as min(f,s) is finite and the sequence is increasing)} \\ &\equiv s \in f(R_i), \text{ for some } i \quad \text{(by the criterion)} \\ &\equiv s \in \bigcup f(R_i). \end{aligned}$$

For strictness if we had $s \in f(\emptyset)$, for some s, then by the criterion $\min(f,s) \subseteq \emptyset$ which would contradict the fact that min(f,s) is nonempty. For multiplicativity, we have:

$$\begin{aligned} s \in f(Q \cap R) &\equiv \min(f,s) \subseteq Q \cap R \equiv \min(f,s) \subseteq Q \text{ and } \min(f,s) \subseteq R \\ &\equiv s \in f(Q) \cap f(R). \quad \boxtimes \end{aligned}$$

Berry has investigated a very general idea of <u>stable</u> continuous functions; our predicate transformers are stable in his sense and Lemma 5.6 is a straightforward consequence of the work in [Ber].

Note that for any predicate transformer, f, the lemma shows that min(f,s) exists and is determined uniquely by the criterion; we now extend the notation when $s \notin f(\text{States})$ by putting $\min(f,s) = \text{States}_\perp$ in that case. This makes $\lambda s \in \text{States}.\ \min(f,s)$ an element of ST, showing how we obtain state transformations from predicate transformers.

<u>Lemma 5.7</u> 1. Let m be in ST. Then $f = \lambda R.\ wp(m,R)$ is in PT and, indeed, for any state, s, $\min(f,s) = m(s)$.

2. Let f be in PT. Then $wp(\lambda s.\min(f,s),R) = f(R)$.

<u>Proof</u> 1. We apply the test of lemma 5.6 to show f is in PT. First suppose $s \in f(\text{States}) = wp(m,\text{States})$. Then $m(s) \neq \perp$, by Definition 5.3 2. and so m(s) is finite and nonempty. Therefore we show $\min(f,s) = m(s)$ by calculating, for any R:

$$\begin{aligned} s \in f(R) \equiv s \in wp(m,R) &\equiv (m(s) \neq \perp) \text{ and } m(s) \subseteq R \\ &\equiv m(s) \subseteq R \text{ (since } m(s) \neq \perp) \end{aligned}$$

2. We just calculate, using lemma 5.6:

$$\begin{aligned} wp(\lambda s.\min(f,s),R) &= \{s \in States \mid \min(f,s) \neq \bot \text{ and } \min(f,s) \subseteq R\} \\ &= \{s \in States \mid s \in f(States) \text{ and } s \in f(R)\} \\ &= f(R). \boxtimes \end{aligned}$$

<u>Definition 5.8</u> The functions ω: ST -> PT and ω^{-1}: PT -> ST are defined by:

$$\omega(m)(R) = wp(m,R)$$
$$\omega^{-1}(f)(s) = \min(f,s).$$

It follows from the above remarks on min(f,s) and lemma 5.7.1. that these are good definitions.

<u>Theorem 5.9</u> <u>(Isomorphism, part I)</u> The function $\omega : ST \cong PT$ is an isomorphism of cpos with two-sided inverse ω^{-1}.

<u>Proof</u> Lemma 5.8 shows that ω and ω^{-1} are mutual inverses as we can calculate: $(\omega^{-1} \circ \omega(m))(s) = \min(\lambda R.wp(m,R),s) = m(s)$ and also: $(\omega \circ \omega^{-1}(f))(R) = wp(\lambda s.\min(f,s),R) = f(R)$.

It remains to show they are monotonic. For ω, suppose $m \sqsubseteq m'$ and $s \in wp(m,R)$. Then $m(s) \neq \bot$ and $m(s) \subseteq R$. So $m'(s) \neq \bot$ and $m'(s) \subseteq m(s) \subseteq R$ (as $m \sqsubseteq m'$!) and so $s \in wp(m',R)$. For ω^{-1} suppose $f \sqsubseteq f'$ and $\min(f,s) \neq States_\bot$. Then using lemma 5.8 we see that $s \in f(\min(f,s)) \subseteq f'(\min(f,s))$ and so $\min(f',s) \subseteq \min(f,s)$ showing that $\min(f,s) \sqsubseteq \min(f',s)$ (order reversal, again!) $\boxtimes$

Wand considered a very slight variant of ST in [Wan]; using our terminology he showed that ω was a bijection. Back independently saw the connection with Smyth's powerdomains in [Bac1] and it is a trivial corollary of his work that ω is monotonic and reflects the partial order on ST; that is for any m, m' in ST,

$$m \sqsubseteq m' \text{ iff } \omega(m) \sqsubseteq \omega(m').$$

Taken together this gives another proof that ω is an isomorphism of cpos.

<u>Lemma 5.10</u> 1. For any p in Bool, m, m' in ST and R in Pred, $wp(\lambda s.$ <u>if</u> $p(s)$ <u>then</u> $m(s)$ <u>else</u> $m'(s),R) = (p^+ \cap wp(m,R)) \cup (p^- \cap wp(m',R))$

2. For any m, m' in ST and R in Pred: $wp(\lambda s.m(s) \cup m'(s),R) = wp(m,R) \cap wp(m',R)$.

3. For any R in Pred, $wp(\bot_{ST},R) = \emptyset$ and $wp(\lambda s.\{s\},R) = R$.

<u>Proof</u> An easy calculation and left to the reader. $\boxtimes$

<u>Lemma 5.11</u> <u>(Homomorphism)</u> 1. <u>Conversion</u> $\forall m \in d\text{-}ST.\ \omega(Conv(m)) = Conv(m)$.
2. <u>Composition</u> $\forall m,m' \in ST.\ \omega(Comp(m,m')) = Comp(\omega(m),\omega(m'))$
3. <u>Conditional</u> $\forall p \in Bool, m \in ST.\ \omega(Cond(p,m)) = Cond(p,\omega(m))$
4. <u>Iteration</u> $\forall p \in Bool, m \in ST.\ \omega(Do(p,m)) = Do(p,\omega(m))$
5. <u>Bar</u> $\forall p,q \in Bool, m,m' \in ST.\ \omega(Bar(\langle p,m\rangle,\langle q,m'\rangle)_1) = Bar(\langle p,\omega(m)\rangle,\langle q,\omega(m')\rangle)_1$
(where for any $p,m, \langle p,m\rangle_1 = m$).

<u>Proof</u> We will often use lemma 5.10 without explicit comment. For any R we

calculate:

1. $\omega(\mathrm{Conv}(m))(R) = wp(\lambda s.\{m(s)\},R)$
$= \{s \mid \bot \in \{m(s)\} \text{ and } \{m(s)\} \subseteq R\}$
$= m^{-1}(R)$
$= \mathrm{Conv}(m)(R),$

2. $\omega(\mathrm{Comp}(m,m'))(R)$
$= wp(\lambda s.\mathrm{App}(m',m(s)),R)$
$= \{s \mid m(s) \neq \bot \text{ and, for all } s' \text{ in } m(s),\ m'(s) \neq \bot \text{ and } \bigcup \{m'(s') \mid s' \in m(s)\} \subseteq R\}$
$= \{s \mid m(s) \neq \bot \text{ and, for all } s' \text{ in } m(s),\ m'(s) \neq \bot \text{ and } m'(s') \subseteq R\}$
$= \{s \mid m(s) \neq \bot \text{ and } m(s) \subseteq wp(m',R)\}$
$= wp(m,wp(m',R))$
$= \mathrm{Comp}(\omega(m),\omega(m'))(R),$

3. $\omega(\mathrm{Cond}(p,m))(R) = wp(\lambda s.\underline{\text{if}}\ p(s)\ \underline{\text{then}}\ m(s)\ \underline{\text{else}}\ \bot, R)$
$= (p^{+} \cap wp(m,R)) \cup (p^{-1} \cap wp(\bot_{ST},R))$ (by lemma 5.10.1.)
$= \mathrm{Cond}(p,\omega(m))(R)$ (by lemma 5.10.3.),

5. $\omega(\mathrm{Bar}(\langle p,m\rangle,\langle q,m'\rangle)_1(R) =$
$(p^{+} \cap [(q^{+} \cap wp(\lambda s.m(s) \cup m'(s),R)) \cup (q^{-} \cap wp(m,R))]) \cup$
$(p^{-} \cap [(q^{+} \cap wp(m',R)) \cup (q^{-} \cap wp(\bot_{ST},R))])$
$= \mathrm{Bar}(\langle p,\omega(m)\rangle,\langle q,\omega(m')\rangle)_1.$ (using lemma 5.10).

4. It is enough to show by induction on k that for all R:

$$wp(\mathrm{Do}_k(p,m),R) = \mathrm{Do}_k(p,\omega(m))(R)$$

For k = 0 both sides are equal to $\emptyset$; for k+1 we calculate:

$wp(\mathrm{Do}_{k+1}(p,m),R) = (p^{+} \cap wp(\mathrm{Comp}(m,\mathrm{Do}_k(p,m)),R)) \cup$
$(p^{-} \cap wp(\lambda s.\{s\},R))$

$= (p^{+} \cap wp(m,wp(\mathrm{Do}_k(p,m),R))) \cup$
$(p^{-} \cap R)$ (by part 2)

$= (p^{+} \cap \omega(m)(\mathrm{Do}_k(p,\omega(m))(R))) \cup$
$(p^{-} \cap R)$ (by induction hypothesis)

$= \mathrm{Do}_{k+1}(p,\omega(m))(R)$. ⊠

Theorem 5.12 (Isomorphism, part II) The isomorphism ω : ST -> PT is also an isomorphism of the semantics, with inverse ω^{-1}, in the sense that the following diagrams all commute:

$$\begin{tikzcd}
\mathrm{Stmts} \arrow[d, "\mathscr{S}_{ST}"'] \arrow[dr, "\mathscr{S}_{PT}"] & \\
\mathrm{ST} \arrow[r, "\omega"'] & \mathrm{PT}
\end{tikzcd}
\qquad
\begin{tikzcd}
\mathrm{GCom} \arrow[d, "\mathscr{G}_{ST}"'] \arrow[dr, "\mathscr{G}_{PT}"] & \\
\mathrm{Bool}\times\mathrm{ST} \arrow[r, "\mathrm{id}_{\mathrm{Bool}}\times\omega"'] & \mathrm{Bool}\times\mathrm{PT}
\end{tikzcd}$$

$$\begin{tikzcd}
\mathrm{Stmts} \arrow[d, "\mathcal{C}_{PT}"'] \arrow[dr, "\mathcal{C}_{ST}"] & \\
PT \arrow[r, "\omega^{-1}"'] & ST
\end{tikzcd}
\qquad\qquad
\begin{tikzcd}
\mathrm{GCom} \arrow[d, "\mathcal{G}_{PT}"'] \arrow[dr, "\mathcal{G}_{ST}"] & \\
\mathrm{Bool}\times PT \arrow[r, "\mathrm{id}_{\mathrm{Bool}}\times\omega^{-1}"'] & \mathrm{Bool}\times ST
\end{tikzcd}$$

where $(\mathrm{id}_{\mathrm{Bool}} \times \omega)(\langle p,m\rangle) =_{\mathrm{def}} \langle p, \omega(m)\rangle$ and $(\mathrm{id}_{\mathrm{Bool}} \times \omega^{-1})(\langle p,f\rangle) =_{\mathrm{def}} \langle p,\omega^{-1}(f)\rangle$.

<u>Proof</u> It follows at once from lemmas 5.10, 5.11, using mutual structural induction on statements and guarded commands that the top two diagrams commute. For example, a typical case is:

$$\begin{aligned}
\text{C4}\quad \omega(\mathcal{C}_{ST}\llbracket(S_1;S_2)\rrbracket) &= \omega(\mathrm{Comp}(\mathcal{C}_{ST}\llbracket S_1\rrbracket, \mathcal{C}_{ST}\llbracket S_2\rrbracket)) \\
&= \mathrm{Comp}(\omega(\mathcal{C}_{ST}\llbracket S_1\rrbracket), \omega(\mathcal{C}_{ST}\llbracket S_2\rrbracket)) \quad \text{(by lemma 5.12)} \\
&= \mathrm{Comp}(\mathcal{C}_{PT}\llbracket S_1\rrbracket, \mathcal{C}_{PT}\llbracket S_2\rrbracket) \quad \text{(by induction hypothesis)} \\
&= \mathcal{C}_{PT}\llbracket(S_1;S_2)\rrbracket.
\end{aligned}$$

It then follows immediately that the second two diagrams commute as, for example:
$\omega^{-1}(\mathcal{C}_{PT}\llbracket S\rrbracket) = \omega^{-1}(\omega(\mathcal{C}_{ST}\llbracket S\rrbracket)) = \mathcal{C}_{ST}\llbracket S\rrbracket$. ∎

Another way to state the theorem is to say that

$\langle \mathrm{id}_{\mathrm{Stmts}}, \mathrm{id}_{\mathrm{GCom}}, \omega, \mathrm{id}_{\mathrm{Bool}} \times \omega\rangle : \langle \mathrm{Stmts}, \mathrm{GCom}, ST, \mathrm{Bool} \times ST\rangle \cong \langle \mathrm{Stmts}, \mathrm{GCom}, PT, \mathrm{Bool} \times PT\rangle$

is an isomorphism of heterogeneous algebras, with inverse,
$\langle \mathrm{id}_{\mathrm{Stmts}}, \mathrm{id}_{\mathrm{GCom}}, \omega^{-1}, \mathrm{id}_{\mathrm{Bool}} \times \omega^{-1}\rangle$. The corresponding part of the work in de Bakker corresponds to showing that ω is a homomorphism: that is that ω: ST -> PT is continuous and the top two diagrams commute.

6. Operational - Denotational

In this final section we tie-up the operational semantics and state transformation function semantics, proving the following theorem.

Theorem 6.1 1. For any statement S and state s, S must terminate from s iff $\mathcal{C}_{ST}[\![S]\!](s) \neq \bot$.

2. If S must terminate from state s then $\mathcal{C}_{ST}[\![S]\!](s) = \{s' \mid \langle S,s\rangle \rightarrow^* s\}$.

The first part of the next corollary shows Dijkstra's direct definition of the predicate transformer semantics is in accord with the operational definition. The second part shows that all the available quasi-orders are the same.

Corollary 6.2 1. For any statement S and predicate R,

$$wp(S,R) = wp(\mathcal{C}_{ST}[\![S]\!],R) = \mathcal{C}_{PT}[\![S]\!](R)$$

2. For any statements S and S',

$$S \underset{\sim}{\sqsubseteq} S' \text{ iff } \mathcal{C}_{ST}[\![S]\!] \sqsubseteq \mathcal{C}_{ST}[\![S']\!] \text{ iff } \mathcal{C}_{PT}[\![S]\!] \sqsubseteq \mathcal{C}_{PT}[\![S']\!]$$

Proof 1. Let s be a state. Then,

$s \in wp(S,R)$ iff S must terminate from s and for any s' in States if $\langle S,s\rangle \rightarrow^* s'$ then s' is in R

iff $\mathcal{C}_{ST}[\![S]\!](s) \neq \bot$ and $\mathcal{C}_{ST}[\![S]\!](s) \subseteq R$ (by Theorem 6.1)

iff $s \in wp(\mathcal{C}_{ST}[\![S]\!],R)$ (by Definition 5.3.2.)

and it follows from Theorem 5.12 that $wp(\mathcal{C}_{ST}[\![S]\!],R) = \mathcal{C}_{PT}[\![S]\!](R)$.

2. We have,

$S \underset{\sim}{\sqsubseteq} S'$ iff $\forall R \in \text{Pred}.\ wp(S,R) \subseteq wp(S',R)$ (by definition)

iff $\forall R \in \text{Pred}.\ \mathcal{C}_{PT}[\![S]\!](R) \subseteq \mathcal{C}_{PT}[\![S']\!](R)$ (by part 1)

iff $\mathcal{C}_{PT}[\![S]\!] \sqsubseteq \mathcal{C}_{PT}[\![S']\!]$

iff $\mathcal{C}_{ST}[\![S]\!] \sqsubseteq \mathcal{C}_{ST}[\![S']\!]$ (by Theorem 5.12). $\boxtimes$

To prove Theorem 6.1 we begin with some useful formulae.

Lemma 6.3 1. For any state s, $\mathcal{C}_{ST}[\![$<u>if</u> <u>empty</u> <u>fi</u>$]\!](s) = \bot$ and $\mathcal{C}_{ST}[\![$<u>do</u> <u>empty</u> <u>od</u>$]\!](s) = \{s\}$.

2. Let G be $B_1 \rightarrow S_1 \,[\!]\, \dots \,[\!]\, B_n \rightarrow S_n$ and let s be a state.

(a) If $\mathcal{B}[\![B_i]\!](s) = ff$ for every i with $1 \leq i \leq n$ then $\mathcal{C}_{ST}[\![$<u>if</u> G <u>fi</u>$]\!](s) = \bot$ and $\mathcal{C}_{ST}[\![$<u>do</u> G <u>od</u>$]\!](s) = \{s\}$.

(b) If $\mathcal{B}[\![B_i]\!](s) = tt$ for some i with $1 \leq i \leq n$ then

$\mathcal{C}_{ST}[\![$<u>if</u> G <u>fi</u>$]\!](s) = \bigcup\{\mathcal{C}_{ST}[\![S_i]\!](s) \mid \mathcal{B}[\![B_i]\!](s) = tt\}$

and $\mathcal{C}_{ST}[\![$<u>do</u> G <u>od</u>$]\!](s) = \bigcup\{\mathcal{C}_{ST}[\![S_i;$<u>do</u> G <u>od</u>$]\!](s) \mid \mathcal{B}[\![B_i]\!](s) = tt\}$.

3. Let S be <u>do</u> $B_1 \rightarrow S_1 \,[\!]\, \dots \,[\!]\, B_n \rightarrow S_n$ <u>od</u>. Then $\mathcal{D}_0[\![S]\!] \sqsubseteq \mathcal{D}_1[\![S]\!] \sqsubseteq$ is an increasing sequence with least upper bound $\mathcal{C}_{ST}[\![S]\!]$ where

the $\mathcal{D}_k[\![S]\!]$ are defined inductively by putting $\mathcal{D}_0[\![S]\!] = \perp$ and for any state s,

$$\mathcal{D}_{k+1}[\![S]\!](s) = \begin{cases} \{s\} & (\text{if } \mathcal{B}[\![B_i]\!](s) = ff \text{ for every } i \text{ with } 1 \leq i \leq n) \\ \bigcup\{\mathrm{Comp}(\mathcal{C}_{ST}[\![B_i]\!], \mathcal{D}_k[\![S]\!])(s) \mid \mathcal{B}[\![B_i]\!](s) = tt\} & (\text{otherwise}) \end{cases}$$

Proof An easy calculation. $\boxtimes$

Now we see that $\mathcal{C}_{ST}$ satisfies a kind of operational fixed-point equation. (The reader may care to improve this to show $\mathcal{C}_{ST}$ is even the least such function of statements.)

Lemma 6.4 If S is not blocked at s then,

$$\mathcal{C}_{ST}[\![S]\!](s) = \{s' \mid \langle S,s\rangle \to s'\} \cup \bigcup \{\mathcal{C}_{ST}[\![S']\!](s') \mid \langle S,s\rangle \to \langle S',s'\rangle\}$$

Proof The proof is by structural induction on S and cases according to its form. When S is an atomic command, skip or abort, the result is trivial. For S = $(S_1;S_2)$ calculate:

$$\begin{aligned}\mathcal{C}_{ST}[\![(S_1;S_2)]\!](s) &= \mathrm{App}(\mathcal{C}_{ST}[\![S_2]\!], \mathcal{C}_{ST}[\![S_1]\!](s)) \\ &= \{\mathcal{C}_{ST}[\![S_2]\!](s') \mid \langle S_1,s\rangle \to s'\} \cup \\ &\quad \bigcup\{\mathrm{App}(\mathcal{C}_{ST}[\![S_2]\!], \mathcal{C}_{ST}[\![S_1']\!](s')) \mid \langle S_1,s\rangle \to \langle S_1',s'\rangle\} \\ &\qquad (\text{by induction hypotheses and Lemma 5.5}) \\ &= \{\mathcal{C}_{ST}[\![S_2]\!](s') \mid \langle S_1,s\rangle \to s'\} \cup \\ &\quad \bigcup\{\mathcal{C}_{ST}[\![(S_1';S_2)]\!](s') \mid \langle S_1,s\rangle \to \langle S_1',s\rangle\} \\ &= \bigcup\{\mathcal{C}_{ST}[\![S']\!](s') \mid \langle(S_1;S_2),s\rangle \to \langle S',s'\rangle\} \quad (\text{consider rule III})\end{aligned}$$

For S = if empty fi blocking occurs; for S = if $B_1 \to S_1 \,[\!]\, \ldots \,[\!]\, B_n \to S_n$ fi if blocking does not occur then $\mathcal{B}[\![B_i]\!](s) = tt$ for some i with $1 \leq i \leq n$ and:

$$\begin{aligned}\mathcal{C}_{ST}[\![S]\!](s) &= \bigcup\{\mathcal{C}_{ST}[\![S_i]\!](s) \mid \mathcal{B}[\![B_i]\!](s) = tt\} \quad (\text{by Lemma 6.3.2.}) \\ &= \bigcup\{\mathcal{C}_{ST}[\![S']\!](s) \mid \langle S,s\rangle \to \langle S',s\rangle\} \quad (\text{consider rule IX})\end{aligned}$$

For S = do empty od the result is trivial from Lemma 6.3.1. For S = do $B_1 \to S_1 \,[\!]\, \ldots \,[\!]\, B_n \to S_n$ od we divide into cases on whether $\mathcal{B}[\![B_i]\!](s) = tt$ for some i and use Lemma 6.3 and consider rule X. $\boxtimes$

The proof of the next lemma uses Noetherian induction (see [Coh], [Hue]). It is also possible to use ordinary induction on the integers by using König's lemma and the finitary nature of the transition relation.

Lemma 6.5 If S must terminate from s then

$$\mathcal{C}_{ST}[\![S]\!](s) = \{s' \mid \langle S,s\rangle \to^* s\}$$

Proof Since S must terminate from s we can define a partial order which satisfies the minimum condition (see [Coh]) on the set $\{C \mid \langle S,s\rangle \to^* C\}$ by putting $C \leq C'$ iff $C' \to^* C$. (That is, $\to^{-1}$ is well-founded on this set.) Then we use Noetherian induction:

$$\mathcal{C}_{ST}[\![S]\!](s) = \{s' \mid \langle S,s\rangle \to s'\} \cup \bigcup \{\mathcal{C}_{ST}[\![S']\!](s') \mid \langle S,s\rangle \to \langle S',s'\rangle\}$$
(by Lemma 6.4)
$$= \{s' \mid \langle S,s\rangle \to s'\} \cup \bigcup \{s'' \mid \exists\, S,s.\langle S,s\rangle \to \langle S',s'\rangle \to^* s''\}$$
(by induction hypothesis)
$$= \{s' \mid \langle S,s\rangle \to^* s'\}.\ \boxtimes$$

Now we only need one more lemma to prove Theorem 6.1.

<u>Lemma 6.6</u> Let S be a statement. Then for any state s if $\mathcal{C}_{ST}[\![S]\!](s) \neq \perp$ then S must terminate from s.

<u>Proof</u> We use structural induction on S and divide into cases according to the form of S. When S is an atomic command, <u>skip</u> or <u>abort</u>, the result is trivial. For S = $(S_1;S_2)$ assume s is a state with $\mathcal{C}_{ST}[\![S]\!](s) \neq \perp$. Then $x = \mathcal{C}[\![S_1]\!](s) \neq \perp$ and for every s' in x, $\mathcal{C}[\![S_2]\!](s') \neq \perp$. Then by induction hypothesis S_1 must terminate from s and, by lemma 6.5 for every s' with $\langle S_1,s\rangle \to^* s'$, $\mathcal{C}[\![S_2]\!](s') \neq \perp$ and so, applying the induction hypothesis again to each such s', S_2 must terminate from s'. Therefore using rule III to consider the possible transition sequences $\langle (S_1;S_2),s\rangle = C_0 \to C_1 \to \ldots.$ we see that S must terminate from s.

For S = <u>if</u> G <u>fi</u>, assume s is a state with $\mathcal{C}_{ST}[\![S]\!](s) \neq \perp$. Then, by lemma 6.3, G cannot be <u>empty</u> but must be of the form $B_1 \to S_1 [\!] \ldots [\!] B_n \to S_n$ with $\mathcal{B}[\![B_i]\!](s) = tt$ for some i. Then applying Lemma 6.3 again, we must have $\mathcal{C}_{ST}[\![S_i]\!](s) \neq \perp$ whenever $\mathcal{B}[\![B_i]\!](s) = tt$ and applying the induction hypothesis we see that for each such S_i that S_i must terminate from s. And so, considering rule IX for the possible transition sequences we see that S must terminate from s.

For S = <u>do</u> G <u>od</u> assume s is a state with $\mathcal{C}_{ST}[\![S]\!](s) \neq \perp$. It follows by lemma 6.3.3. that $\mathcal{D}_k[\![S]\!](s) \neq \perp$ for some k. We finish the proof by showing by induction on k that for any state s if $\mathcal{D}_k[\![S]\!](s) \neq \perp$ then S must terminate from s. This is trivial for k = 0; for k+1 assume that s is a state such that $\mathcal{D}_{k+1}[\![S]\!](s) \neq \perp$. The case where G is <u>empty</u> is trivial and we can assume G is $B_1 \to S_1 [\!] \ldots [\!] B_n \to S_n$. If $\mathcal{B}[\![B_i]\!](s) = ff$ for every i with $1 \leq i \leq n$ then the result is trivial; otherwise by lemma 6.3.3, we have $\mathrm{Comp}(\mathcal{C}_{ST}[\![S_i]\!], \mathcal{D}_k[\![S]\!])(s) \neq \perp$ for every i such that $\mathcal{B}[\![B_i]\!](s) = tt$. Then using the induction hypothesis and applying the same argument as in the case where S had the form $(S_1;S_2)$ we see that $(S_i;S)$ must terminate from s and it follows at once that S too must terminate from s, concluding the proof by induction. $\boxtimes$

<u>Proof of Theorem 6.1</u> Part 2 is just Lemma 6.5. For part 1 the necessity half is immediate from part 2 and sufficiency is just Lemma 6.6. $\boxtimes$

References

[ADJ] Goguen, J.A., Thatcher, J.W., Wagner, E.G. and Wright, J.B. (1977) Initial Algebra Semantics and Continuous Algebras. JACM, Vol. 24, No. 1, 68-95.

[Bac1] Back, R-J. (1979) On the notion of correct refinement of programs. Department of Computer Science, University of Helsinki.

[Bac2] Back, R-J. (1979) Proving Total Correctness of Nondeterministic Programs in Infinitary Logic. Department of Computer Science, University of Helsinki.

[Ber] Berry, G. (1978) Stable Models of Typed λ-Calculi. Proc. of 5th ICALP. Lecture Notes in Computer Science, Vol. 62 (eds. G. Ausiello and C. Böhm) Berlin: Springer-Verlag.

[Coh] Cohn, P.M. (1965) Universal Algebra. New York: Harper and Row.

[de B] de Bakker, J.W. (1978) Recursive Programs as Predicate Transformers. Formal Description of Programming Concepts (ed. E.J. Neuhold) Amsterdam: North Holland.

[de R] de Rover, W.P. (1976) Dijkstra's Predicate Transformer, Non-Determinism Recursion and Termination. Proc. 5th Symposium Mathematical Foundations of Computer Science (ed. A. Mazurkiewicz) pp. 472-481. Lecture Notes in Computer Science, Vol. 45, Berlin: Springer-Verlag.

[Dij] Dijkstra, E.W. (1976) A Discipline of Programming. New Jersey: Prentice-Hall.

[Gor] Gordon, M.J.C. (1979) The Denotational Description of Programming Languages. An Introduction. Berlin: Springer-Verlag.

[Har] Harel, D. and Pratt, V.R. (1978) Nondeterminism in Logics of Programs. Proc. of the 5th Ann. ACM Symposium on Principles of Programming Languages, Tucson, Arizona, pp. 203-213.

[Hen] Hennessy, M.C.B. and Plotkin, G.D. (1979) Full Abstraction for a Simple Parallel Programming Language. Proc. of the 8th Symposium on Mathematical Foundations of Computer Science. Lecture Notes in Computer Science, No. 74 (ed. J. Becvar) pp. 108-120, Berlin: Springer-Verlag.

[Hue] Huet, G. (1977) Confluent Reductions: Abstract Properties and Applications to Term Rewriting Systems. Proc. of the 18th IEEE Symposium on Foundations of Computer Science, pp. 30-45.

[Jen] Jensen, K. (1978) Connection between Dijkstra's Predicate Transformers and Denotational Continuation Semantics. DAIMI PB-86, Dept. of Computer Science, University of Aarhus.

[Kel] Keller, R.M. (1976) Formal Verification of Parallel Programs. Comm. ACM 19, 7, 371-384.

[Mil1] Milne, R.E. and Strachey, C. (1976) A Theory of Programming Language Semantics. New York: Wiley.

[Mil2] Milne, R.E. (1978) Transforming Predicate Transformers. Formal Description of Programming Concepts (ed. E.J. Neuhold) Amsterdam: North Holland.

[Mil3] Milner, R. (1976) Program Semantics and Mechanised Proof. Foundation of Computer Science II (eds. K.R. Apt and J.W. de Bakker) Mathematical Centre Tracts 82, Mathematisch Centrum, Amsterdam.

[Plo1] Plotkin, G.D. (1975) Call-by-Name, Call-by-Value and the λ-Calculus. TCS, Vol. 1, 125-159.

[Plo2] Plotkin, G.D. (1976) A Powerdomain Construction. SIAM Journal on Computation, Vol. 5, No. 3, 452-487.

[Smy] Smyth, M. (1978) Powerdomains. JCSS, Vol. 16, No. 1.

[Sto] Stoy, J.E. (1977) Denotational Semantics: the Scott-Strachey Approach to Programming Language Theory. MIT Press.

[Str] Strachey, C. and Wadsworth, C.P. (1974) Continuations. A mathematical semantics for handling full jumps. Technical Monograph PRG-11, Oxford University Computing Laboratory.

[Wad] Wadwworth, C.P. (1980) Semantic Domains for Programming Languages. Prentice-Hall. To appear.

[Wan] Wand, M. (1977) A Characterisation of Weakest Preconditions. JCSS, Vol. 15, No. 2, 209-212.