数理解析研究所講究録
第 454 巻 1982 年 211-228

A PREDICATE TRANSFORMER FOR WEAK FAIR ITERATIO

BY

DAVID PARK

(to appear in : Proceedings, 6th IBM Symposium on Mathematical Foundations of Computer Science, Hakone, Japan.)

Department of Computer Science,
University of Warwick,
COVENTRY CV4 7AL,
ENGLAND.

May 1981

A Predicate Transformer for Weak Fair Iteration

David Park
Department of Computer Science
University of Warwick
Coventry CV4 7AL

Abstract: Two new constructs, wdo- and sdo- statements, are added to the language of guarded commands as varieties of do-statement, with fairness constraints on infinite execution sequences. With weak fairness, a clause must be executed infinitely often unless its guard is infinitely often false; with strong fairness, unless its guard is only finitely often true. The relationship to unbounded nondeterminism is discussed, and a predicate transformer wp (WDO,R) obtained for the weak version, by introduction of fixpoint concepts.

Introduction:

In the author's previous work [9], [10], fairness was considered in the context of parallelism, as in combinations such as:

$$(\text{while } b \text{ do skip}) \text{ par } b := \text{false}$$

The emphasis there was to capture the constraint on all interleavings of program executions, that all steps get executed ultimately - the constraint that guarantees termination of the example above.

This paper studies fairness in a slightly different setting, by considering constraints on the execution of guarded iterations

$$\text{do } B1 \rightarrow C1 \;\square\; B2 \rightarrow C2 \;\square\; \ldots \square\; Bn \rightarrow Cn \text{ od}$$

which are appropriate when these are regarded as controlling the parallel execution of n processes each consisting of the iteration of some Ci. In this context, one looks for conditions to guarantee termination of

$$\text{do } b \rightarrow \text{skip} \;\square\; b \rightarrow b := \text{false} \text{ ~~end~~ od}$$

The work here has been inspired particularly from three recent sources. The setting, of considering fair iteration constraints, is taken directly from Apt & Olderog [1]; the statement transformations T(WDO), T(SDO) of Section 3 are simplifications of transformations first described by them (I was previously in doubt whether any such T(SDO) existed). The predicate transformer of Section 4 arose from considering a preliminary version of Lehmann, Pnueli & Stavi [6]; the corollary, completeness of a simple rule for weak fair termination (called "justice" by them), was arrived at independently by myself - though the inspiration clearly owes much to their own original characterization. The predicate transformer itself derives from preliminary work by de Roever aimed at capturing strong fairness in the mu calculus of [5]. I am indebted to the Bad Honnef Workshop on Semantics, where these ideas germinated, and to the Programming Research Group, Oxford, where they were developed.

1. Preliminaries:

Language: The programming language to be considered will be an extension of Dijkstra's language of guarded commands as presented in [4]. Alternative forms wdo, sdo for do will be described presently. In addition, we add a "nondeterministic" expression

$$?$$

which is to take as value any nonnegative integer on each evaluation, not necessarily the same integer at different occurrences or evaluations. Thus (99 + ?) may take on any integer value 99 or above; [in fact (? - ?) takes on any integer value, and (? = ?) has values true and false - though we will not here be obsessed with such issues]

Predicates: For uniformity with [4] (and also, for variety) we will cast our ideas in terms of predicates rather than sets. While this implies a widespread recasting of old terminology, the differences are here regarded as cosmetic rather than essential. Taking S to be the set of states s, we will talk of $s \models A$ rather than $s \in A$, and of $A \wedge B$ rather than $A \cap B$. More problematically, we will use infinite combinations such as $\bigvee_{\lambda<\alpha} F_\lambda$, for transfinite α and for an indexed set $\{F_\lambda\}$ of predicates, without concern for expressibility or effectiveness of the result. Insofar as predicates are regarded as having syntax, it will be the syntax used for forming guards (but not involving the nondeterministic "?" expression) extended by adding conventional logical combinations and the $\Box\!-$, $\Diamond\!-$ operators introduced below. Finally, we will frequently assert a predicate A, where strictly speaking we should write that $s \models A$ for all s.

Semantics: The semantics of the language as specified by axioms for the wp predicate transformer is usually distinguished from its denotational semantics. But there is a close relationship between wp-semantics and a relational denotational semantics in the style of [9]. There, a denotational semantics was obtained for a language involving parallelism and unbounded nondeterminism, by specifying two semantic functions on programs C

$$M(C) \subseteq S \times S, \text{ the relation computed by C}$$
$$\mathcal{T}(C) \subseteq S, \quad \text{the termination domain of C}$$

From M(C) we can define the operators $[C]$, $\langle C\rangle$ of dynamic logic, which are convenient predicate transformers to use in conjunction with wp. The relationships are as follows:

$$s \models [C]R \quad \text{iff for all } s', \langle s,s'\rangle \in M(C),\ s' \models R$$
$$s \models \langle C\rangle R \quad \text{iff there exists } s', \langle s,s'\rangle \in M(C) \text{ and } s' \models R$$
$$wp(C,R) \equiv \mathcal{T}(C) \wedge [C]R$$

In this paper we are above all concerned with guaranteed termination, which is

$$wp(C, \underline{true}) \equiv \mathcal{T}(C)$$

since $[C]\ \underline{true} \equiv \underline{true}$ for all C.

In a wider context, termination properties generalise to the "liveness" properties discussed by Owicki & Lamport [7], as distinguished from "safety" properties which are related to our predicates $[C]R$.

Fixpoints: to obtain a wp(DO,R) which is correct when there is unbounded nondeterminism, and for our wp(WDO,R) we need to translate fixpoint theory into "predicate-theoretic" terms:

Definition: A predicate transformation F(X) is monotone in X if, whenever $X \Rightarrow Y$ (i.e. whenever $s \models (X \Rightarrow Y)$, all s) then $F(X) \Rightarrow F(Y)$.

Theorem [Knaster-Tarski]: If F(X) is monotone, the identity

$$X \equiv F(X)$$

has a strongest solution (the least fixpoint μF) and a weakest solution (the maximal fixpoint νF).

Note: This assumes that "predicates" form a complete boolean algebra - a somewhat artificial assumption if "predicates" are to be construed as expressible in some fixed formalism.

Fixpoint Induction: The following are valid inferences:

(F.I.) from $F(X) \Rightarrow X$ to deduce $\mu F \Rightarrow X$

(dual F.I.) from $X \Rightarrow F(X)$ to deduce $X \Rightarrow \nu F$

Proofs of these results are well-known (see [8]).

2. Unbounded Nondeterminism:

There are general conceptual problems to do with unbounded nondeterminism; the reader may be familiar with the arguments in Chapter 9 of Dijkstra [4]. In this section we summarise the relevant portions of [9]. In particular we will reformulate Dijkstra's characterization of wp(DO,R) so as to make unbounded nondeterminism acceptable (prior to [9], this formulation was made in Boom [3]).

Fairness constraints ensure termination of statements such as:

b := true; z := 0;
wdo b → z := z+1 □ b → b := false od

But they put no bound on the resulting value of z.

This means that the above would be equivalent to (i.e. would map to the same objects under M(C), T(C) as)

b : = false; z := ? ;

The fairness constructs therefore face similar objections to those concerning unbounded nondeterminism , since expressions involving " ? " may be systematically eliminated in favour of wdo (or sdo). There are three points to be dealt with:

2.1 Iterations and Unbounded Nondeterminism

There is a definitional problem, illustrated by the example

C : do z < 0 → z := ? □ z > 0 → z := z-1 od

in which z is taken to vary over integers only. According to Dijkstra's specification, we should have

$$T(C) \equiv wp(C, \underline{true}) \equiv \bigvee_{i=0}^{\infty} H_i$$

where H_i is defined inductively by

$$H_0 \equiv (z = 0)$$

$$H_{i+1} \equiv H_i \vee ((z < 0 \Rightarrow wp(z := ?, H_i)) \wedge (z > 0 \Rightarrow wp(z := z-1, H_i))) \qquad (*)$$

which solves as

$$H_i \equiv (0 \leqslant z \leqslant i)$$

noting that

$$wp(z := ?,\ 0 \leqslant z \leqslant i) \equiv \underline{false}$$
$$wp(z := z-1,\ 0 \leqslant z < i) \equiv 0 < z \leqslant i+1$$

But then

$$T(C) \equiv \bigvee H_i \equiv (z \geqslant 0)$$

This is anomalous. We expect $T(C) \equiv \underline{true}$.

The anomaly disappears if the definition (*) is reformulated in fixpoint terms.

$$\left.\begin{array}{l} T(C) \equiv H \text{ where H is the } \underline{\text{strongest}}\ \underline{\text{solution}} \text{ to} \\ \quad H \equiv (z < 0 \Rightarrow wp(z := ?,\ H)) \\ \quad\quad \wedge (z > 0 \Rightarrow wp(z := z-1,\ H)) \end{array}\right\} \quad (**)$$

(**) now determines $T(C) \equiv \underline{true}$; $H \equiv \underline{true}$ is in fact the unique solution to the equivalence.

An amended definition for wp can now be given to allow for unbounded nondeterminism. As in [4], DO denotes the statement

$$\underline{do}\ \ldots\ldots\ \square\ B_i \rightarrow C_i\ \square\ \ldots\ldots\ \underline{od}$$

Replace the wp(DO,R) definition by:

2.0.1. wp(DO,R) is the strongest solution H to $H \equiv (\bigwedge_i \neg B_i \Rightarrow R) \wedge \bigwedge_i (B_i \Rightarrow wp(C_i, H))$

2.0.1 can be justified, incidentally, by appeal to an operational semantics as we will do in Section 4 for WDO. If the right hand side of the equation in 2.0.1. is abbreviated F(H), the scheme is to show

(i) $F(wp(DO,R)) \Rightarrow wp(DO,R)$; so $\mu F \Rightarrow wp(DO,R)$ by Fixpoint Induction

(ii) If $H \equiv F(H)$ and $s \models \neg H$
then $s \models \neg\, wp(DO,R)$;
so $wp(DO,R) \Rightarrow \mu F$.

But we omit the details of this scheme. We should note the case when $R \equiv \underline{true}$ and each C_i terminates:

2.0.2. If $B_i \Rightarrow T(C_i)$ then T(DO) is the strongest solution to

$$H \equiv \bigwedge_i (B_i \Rightarrow \boxed{C_i}\ H)$$

Finally, we should cope with the addition of " ? ". It will suffice to restrict "?" to the right hand sides of assignments only. Then we must add:

2.0.3. If E contains occurrences of " ? ", then

$$wp(x := E,R) \equiv \bigwedge_{E' \in d(D)} wp(x := E',R)$$

where d(E) is the set of expressions obtainable from E by substitution of nonnegative numerals for occurrences of " ? ".

2.2 Continuity

The anomaly in 2.1 is that $\bigvee H_i$ does not necessarily satisfy the fixpoint identity

$$H \equiv \bigwedge_i (B_i \Rightarrow wp(C_i,H))$$

This indicates a failure of continuity. In the particular example the right-hand side F(H) is not a continuous function of H. We have

$$F(\bigvee_{i=o}^{\infty} H_i) \not\equiv \bigvee_{i=o}^{\infty} F(H_i)$$

for a sequence with $H_i \Rightarrow H_{i+1}$, all i.

A simpler example of a non-continuous function is just

$$F(H) \equiv wp(x := ?, H)$$

Taking $H_i \equiv x < i$, we have $F(H_i) \equiv$ false for all i; but $\bigvee H_i \equiv$ true. So

$$F(\bigvee H_i) \equiv wp(x := ?, \text{true}) \equiv \text{true} \not\equiv \bigvee F(H_i)$$

While monotonicity can replace continuity for the purposes of [4], failure of continuity forces a departure from the usual assumptions of domain-theoretic denotational semantics (as described in Stoy [11], for example). This is a deep technical difficulty, which motivated the return to elementary relational notions of Park [9]. Apt & Plotkin [2] present recent work aimed at reconciling the domain-theoretic approach.

2.3 Implementability

The continuity constraint which is violated in 2.2 is one which arises from a priori considerations of what is "computable". We should expect an anomaly in this respect also. For example, consider the (diverging) statement

<u>do</u> z=0 → z:=?+1; b:=¬b □z>0 → z:=z−1; <u>if</u> b → write (0) □ b → write (1) <u>fi</u> <u>od</u>

This should produce as output an infinite sequence over {0,1} - and the set of all possible outputs forms the "fair set"

$$(0^*1\ 1^*0)^{\omega}$$

of all sequences with infinite numbers of both 0s and 1s. This set is not "computable" in any accepted sense; testing any finite number of initial segments of a sequence is irrelevant to deciding membership in the set.

The technical consequences of this difficulty are essentially those mentioned in 2.2. But there is an added perplexity. If the denotation is "uncomputable", it seems to follow that the program is "unimplementable".

We have to consider what "implementation" means in the sense of nondeterminism being used.

<u>Definition</u>: A statement C1 is a <u>slice</u> of statement C2 iff wp(C2,R) ⇒ wp(C1,R) for all predicates R.

Thus, x := 1 is a slice of x := ?. If we admit as "implementations" of C2 any implementation of a slice C1, the perplexity disappears (presumably C1 is "minimal" - i.e. a deterministic slice). The slice C1 may be "computable" independent of C2. The language specifier is not interested in the enumerability of all possible results of "nondeterministic" programs, in the sense in which he uses the term. If he does intend some such "tight" sense of nondeterminism, he will need to distinguish it from the "loose" sense we are used to hearing from him.

2.4 Termination and Ordinals

The reader familiar with ordinals, transfinite induction, and their connection with fixpoints of non-continuous functions will be interested in the link between the termination predicate 2.0.2. and the familiar

> 2.4.1. If $(B_i \Rightarrow \mathcal{T}(C_i))$ then $s \models \mathcal{T}(DO)$ iff there exists a well-ordering (W,>) and a partial map f : S → W with
>
> (i) f(s') defined, $\langle s',s''\rangle \in M(C)$ imply f(s") defined, f(s') > f(s")
>
> (ii) f(s) is defined.

Adequacy of this rule is obvious. An infinite iteration would produce an infinite descending sequence in W, contra well-foundedness.

Definition: Given a function F(X) on predicates, define F^λ for ordinals λ, by

$$F^o \equiv \underline{false}$$
$$F^\alpha \equiv F(\bigvee_{\lambda<\alpha} F^\lambda)$$

Theorem 2.4.2: The strongest solution to

$$X \equiv F(X)$$

F monotone, is F^α , some ordinal α.

[for proof, see [5]]

2.4.1 can then be shown to be complete by taking F(H) as the right hand side in 2.0.2., a monotone combination of H; taking W = α from 2.4.2., and defining

$$f(s) = \min \{\lambda \ / \ s \models F^\lambda\}$$

In the case that F is continuous, we can take $\alpha = \omega$, so that only finite ordinals need be involved. But if unbounded nondeterminism occurs, ordinals up to ω^ω may be necessary (see [1]).

3. Strong and weak fairness

"Fairness" is to be expected when

$$DO : \underline{do} \ \ldots \ \square \ B_i \rightarrow C_i \ \square \ldots \ \underline{od}$$

is thought of as scheduling processes in parallel by suitable interleaving, the i^{th} process being just the iteration of C_i. If processes do not interfere with each other, i.e. if no C_i affects B_j, $j \neq i$, the appropriate fairness interior is clear. With interference there is a choice.

Notation: Write C_i: $s \mapsto s'$ for $<s,s'> \in M(C_i)$ & $s \models B_i$

Definitions: A finite or infinite sequence $s_o s_1$ ... is a DO-sequence if, for each $i \neq 0$, there is a j such that $s_{i-1} \models B_j$ and $C_j : s_{i-1} \mapsto S_i$

[DO-sequences provide the operational notion needed for checking 2.0.1. in the way indicated.]

An infinite sequence is <u>weak fair</u> if, for each j,

either (i) $C_j : s_{i-1} \mapsto s_i$ for infinitely many i

or (ii) $s_i \models \neg B_j$ for infinitely many i.

The sequence is <u>strong fair</u> if, for each j,

either (i) above

or (ii) $s_i \models B_j$ for only finitely many i.

The fairness constraints allow us to disregard some infinite DO-sequences. To indicate contexts where only weak fair infinite sequences are considered, we replace <u>do</u> by <u>wdo</u>; or by <u>sdo</u> if only strong fair sequences are considered.

The effect of replacing <u>do</u> by <u>wdo</u>, or <u>wdo</u> by <u>sdo</u> is to increase T(DO), without affecting the relation M(DO). As an example, the statements

<u>wdo</u> b → <u>skip</u> □ b → b := <u>false</u> <u>od</u>

<u>sdo</u> b2 → b1 := ¬b1

□ b1 → b2 := <u>false</u>; b1 := <u>false</u> <u>od</u>

both have guaranteed termination, though neither need terminate with weaker constraints. In the first case, infinite repetition of the <u>skip</u> is not weak fair, since the other guard would remain continuously true. In the second case, infinite repetition of b1 := ¬b1 is weak but not strong fair. b1 is infinitely often true and infinitely often false.

The weak/strong terminology derives from Apt, Plotkin & Olderog (see [1]). Strong fairness is favoured in the literature - a point we return to in a moment. Lehmann, Pnueli & Stavi [6] refer to weak fairness as "justice".

3.1 Fairness and Unbounded Nondeterminism

In Section 2 we pointed out how unbounded nondeterminism can be simulated using <u>wdo</u> or <u>sdo</u>. Apt & Olderog [1] prove a converse result, that both varieties of fair iteration can be simulated, using statements involving " ? " and ordinary <u>do</u>-loops. Here we will present rather simpler versions of the transformations used by them and proofs using informal operational reasoning [the reader should be wary of accepting plausible alternative transformations without rigorous proof.] Since the <u>wdo</u> justification suffers from a technical complication, we consider the <u>sdo</u> transformation first.

3.1.1. strong fairness

SDO : sdo ...... $\square\ B_i \rightarrow C_i\ \square$...... od

T(SDO) : .... z_i := ? ; ...... ;

do ......

$\square\ B_i \wedge \bigwedge_j (B_j \Rightarrow z_i \leqslant z_j) \rightarrow C_i;\ z_i := z_i + 1 + ?$

$\square$ ......

od

To see that SDO, T(SDO) are equivalent (disregarding the introduction of z_i), consider first any finite or strong fair infinite SDO-sequence $s_o s_1$ .... We need to arrange successive values for " ? " so that T(SDO) goes through the corresponding sequence of clauses. This can be done by arranging that z_i, at the j^{th} step, holds either

(i) $\min \{k \mid k \geqslant j,\ C_k : s_k \mapsto s_{k+1}\}$

or (ii) if there is no such k, some k such that $m \geqslant k \Rightarrow s_m \models \neg B_i$ for all m

Such a choice is possible, at any step, since the SDO-sequence is either strong fair or finite (in which case it terminates with some $s_k \models \bigwedge \neg B_i$). In the converse direction, there is no difficulty in seeing that every T(SDO)-sequence corresponds to an SDO-sequence; every iteration obeys an appropriate guarded command. To see that infinite sequences are strong fair, note that C_i is executed only finitely often if and only if z_i reaches some final value. After some N iterations, each such z_i will have reached its final value, and every other z_j will bound all such final values. But then each such B_i must remain false in all later iterations.

3.1.2. weak fairness

WDO : wdo ...... $\square\ B_i \rightarrow C_i\ \square$ ...... od

T(WDO) : ...... ; z_i := ? ......

do ......

$\square \bigwedge_j (z_i \leqslant z_j) \rightarrow$ if $B_i \rightarrow C_i\ \square\ \neg B_i \rightarrow$ skip fi;
$z_i := z_i + 1 + ?$

$\square$ ......

od

$\bigvee_j B_j \wedge$

Proceeding as in 3.1.1., the complication arises in simulating a particular WDO-sequence $s_o s_1 \ldots$. We need to cope with "dead" clauses, which WDO no longer takes, but which are still periodically inspected by T(WDO). One solution is to encode the death/life property in z_i as follows: when the minimal z_k is 2j or 2j+1, T(WDO) is about to simulate the transition from s_j to s_{j+1}. The index z_i is then

either (i) $2k+1$ where $k = \min \{m/C_i : s_m \longmapsto s_{m+1},\ m \geq j\}$

or (ii) $2k$, if C_i is dead (i.e. if the set in (i) is empty) for some $k \geq j$, with $s_k \models \neg B_i$.

An appropriate value for z_i here always exists, from weak fairness, or from the termination condition, if the WDO-sequence is finite.

The converse direction for 3.1.2. is not difficult. Every iteration of T(WDO) either skips or obeys an appropriate guarded command. To check weak fairness of infinite computations, note that T(WDO) executes each clause infinitely often; so if some C_i is executed only finitely often, the corresponding B_i must be false infinitely often.

3.2 Simulating strong fairness with weak fairness

There is a direct construction, as follows:

Tw(SDO) : ... ; b_i := false; ...
... ; z_i := 0 ; ...
wdo ......
□ $\bigwedge_j ((B_j \wedge b_j) \Rightarrow z_i \leq z_j) \wedge \bigvee_j B_j$
→ if $B_i \rightarrow C_i$; $z_i := z_i+1$; b_i := false
□ $\neg B_i \rightarrow b_i$:= true fi
□ ......
od

Justification: An SDO-sequence is simulated by the Tw(SDO)-sequence which makes corresponding clause choices; but C_i may be chosen at any point from which B_i will remain false, to ensure weak fairness. In the converse direction, consider those clauses for which b_i holds as forming a "queue" which is ordered by the corresponding z_i, which holds the number of times C_i has been executed. At each step, either both loops terminate, or at least one guard of Tw(SDO) is true - for the earliest clause in the queue whose guard holds (or for every clause, if no guard in the queue holds). Moreover at any stage, Tw(SDO) eventually executes some C_i, after possible additions to the queue, either by selecting the earliest appropriate clause (which is eventually done, by weak fairness) or otherwise. To see that weak fair Tw(SDO)-sequences correspond to strong fair SDO-sequences, the argument proceeds as for T(SDO). In some final segment of the computation, the values of z_i for C_i obeyed finitely often remain constant, bounded by the other values z_j. Among these dead clauses, those in the queue must have guards which remain false, or else the guard in Tw(SDO) for the earliest such would be the only true guard and it would be resurrected. But this means the Tw(SDO)-guard for any dead clause (in the queue or not) must be true throughout the final segment; so the corresponding command eventually gets obeyed, which must add it to the queue, and its guard must be false thereafter. So the only clauses of SDO executed finitely often have guards which are true only finitely often.

3.3 Implementing fairness

Here we are interested in efficient slices of WDO, SDO.

3.3.1. weak fairness

There are many reasonably efficient slices - what is involved is a straightforward fair scheduling algorithm, for example:

$z := 1$

do ......

$\Box \bigvee_j B_j \wedge$ $z = i \rightarrow$ **if** $B_i \rightarrow C_i \;\Box\; \neg B_i \rightarrow$ **skip fi**; $z := i + 1$

.....

$\Box \bigvee_j B_j \wedge$ $z = n \rightarrow$ . . . . ; $z := 1$

od

3.3.2. strong fairness

One algorithm is obtained from T(SDO) above, by suitable choice of " ? " :

replace z_i := ? by $z_i := 0$

$z_i := z_i + ? + 1$ by $z_i := \max_j\{z_j\} + 1$.

This is, in effect, a "queueing" algorithm; at each stage, the earliest clause with true guard is obeyed, and moved to the end of the queue.

3.3.3. Discussion

The problem of implementing strong fairness is disquieting. It is not clear that there is any algorithm which is essentially more efficient than the queueing algorithm of 3.3.2. All that have been explored by this author involve the (eventual) memorization of arbitrary queue states (i.e. of arbitrary permutations of {1,2 ... n}) and the overheads of excising from and adding to queues. If the problem is essentially as complex as this, then strong fairness in this form would seem an undesirable ingredient of language specification, at least as the sole "low level" fair primitive. As we have shown in 3.2, strong fairness can be simulated using weak fairness with a queueing regime. One suspects that programmers would prefer the option of coding some such regime - and exploiting special features of their task to simplify it. If strong fairness is the only fair construct around, and has inherent inefficiency of the order suspected, there is cause for concern on pragmatic grounds. One waits for appropriate complexity results.

4. A Weakest Precondition for WDO

We will establish and justify wp(WDO,R).

Notation: EA(WDO,A,Y) is the weakest solution to

$$X \equiv A \wedge \bigwedge_i (B_i \Rightarrow wp(C_i,\ X \vee Y))$$

Lemma 4.1: $s \models EA(WDO,A,Y)$ iff $s \models A$, and for every WDO-sequence $s_o s_1 \ldots$ with $s = s_o$,

either (i) $s_i \models Y$ for some $i > 0$

or (ii) $s_i \models A$ ~~for all $i \geq 0$~~ $\wedge\ (B_j \Rightarrow T(C_j))$ for all $j \geq 0$.

Proof: ⇒) let $H \equiv EA(WDO,A,Y)$ and let $s_o s_1 \ldots$ be a WDO-sequence with $s_i \models \neg Y$, $i > 0$, $s_o \models H$. If $s_i \models H$ then $s_i \models B_j \Rightarrow wp(C_j,\ H \vee Y)$ for each j, from the fixpoint equation for EA(WDO,A,Y). So $s_{i+1} \models H \vee Y$; so $s_{i+1} \models H$ since $s_{i+1} \models \neg Y$ by choice. So every $s_i \models H$, and $H \Rightarrow A$ from the fixpoint equation.

⇒) Let $\hat{H}$ abbreviate the converse predicate of s. We use the dual Fixpoint Induction principle, by showing

$$\hat{H} \Rightarrow A \wedge \bigwedge_i (B_i \Rightarrow wp(C_i,\ \hat{H} \vee Y)).$$

Clearly, $\hat{H} \Rightarrow A$. Suppose

$$s = s_o \models \hat{H} \wedge A \wedge B_i \wedge \neg wp(C_i,\ \hat{H} \vee Y).$$

Then there exists s_1 such that $C_i : s_o \mapsto s_1$, and $s_1 \models \neg(\hat{H} \vee Y)$. Since $s_1 \models \neg\hat{H}$, there is a WDO-sequence $s_1 s_2 \ldots s_n$ with $s_i \models \neg Y$, $i \geq 0$ and $s_n \models \neg A$. But then $s_o s_1 \ldots s_n$ contradicts $s_o \models \hat{H}$.

Corollary: EA(WDO,A,Y) is monotone in A,Y. [Since the equivalent predicate is clearly monotone in A,Y.] The required wp is now

4.2

wp(WDO,R) is the strongest solution H

to

$$H \equiv (\bigwedge_i \neg B_i \wedge R) \vee \bigvee_i EA(WDO, B_i \wedge wp(C_i,H), H)$$

Note that wp(WDO,R) is defined using alternating fixpoints, since EA involves the weakest solution to a fixpoint equation.

Justification of 4.2:

Let $\hat{H}$ be wp(WDO,R) as defined operationally - the weakest predicate guaranteeing termination with R. We must prove $\hat{H} \equiv H$, where H is defined in 4.2.

$\Rightarrow$) Use standard Fixpoint Induction; we must show

$$(\bigwedge \neg B_i \wedge R) \vee \bigvee_i EA(WDO, B_i \wedge wp(C_i,\hat{H}), \hat{H}) \Rightarrow \hat{H}$$

Clearly $(\bigwedge \neg B_i \wedge R) \Rightarrow \hat{H}$; suppose

$$s_o \models EA(WDO, B_i \wedge wp(C_i,\hat{H}), \hat{H})$$

From the Lemma, if a WDO-sequence avoids $\hat{H}$, it passes only through states $s_j \models B_i \wedge wp(C_i,\hat{H})$. But in such a sequence C_i would remain permitted but never applied; so the sequence would be infinite and not weak fair. So all finite or weak fair sequences reach $\hat{H}$; so $s_o \models \hat{H}$, since termination with R is guaranteed.

$\Rightarrow$) suppose $s_o \models \neg H$; we construct an embarrassing WDO-sequence $s_o \ldots s_1 \ldots s_2 \ldots$ with each $s_i \models \neg H$, by induction on i. Suppose $s_i \models \neg H$, then

$$s_i \models \neg EA(WDO, B_j \wedge wp(C_j,H), H)$$

for each j. So for each j, from the Lemma, we can find a WDO-sequence to some

$$s' \models \neg H \text{ with } s' \models \neg(B_j \wedge wp(C_j,H)).$$

If $s' \models \neg B_j$, take $s_{i+1} = s'$; otherwise choose $C_j : s' \mapsto s_{i+1}$ with $s_{i+1} \models \neg H$. This can be repeated indefinitely, for any sequence of clauses C_j. By choosing each j infinitely often we obtain $s_o, s_1 \ldots$ such that either (i) there is an infinite weak fair WDO-sequence through $s_o, s_1 \ldots$

or (ii) some $s_i \models \bigwedge \neg B_i \wedge \neg R$, and there is a finite WDO-sequence reaching $\neg R$.

Finally, we can obtain our analogue of the Lehmann, Pnueli & Stavi result [6] .

<u>Corollary 4.3</u>:

If $B_i \Rightarrow T(C_i)$, then $s \models T(\mathrm{WDO})$ iff there exist a well ordering $(W, >)$, a partial map $f : s \to W$, and predicates Q, Q_i with

(i) $s \models Q$ iff $s \models \bigvee_i Q_i$ iff $f(s)$ is defined

(ii) if $s \models Q$, $C_i : s \mapsto s'$ then $s' \models Q$ and $f(s) \geqslant f(s')$

(iii) if $s \models Q_i$, $C_j : s \mapsto s'$, $f(s) = f(s')$ then $s' \models Q_i$

(iv) if $s \models Q_i$, $C_i : s \mapsto s'$ then $f(s) > f(s')$

(v) $Q_i \Rightarrow B_i \vee \bigwedge_j \neg B_j$

<u>Proof</u>: analogous to the argument in 2.4. Suppose (i) - (v) are satisfied for s, and $s_o s_1 \ldots$ is an infinite WDO-sequence from $s = s_o$. From (ii) every $f(s_i)$ is defined, and $f(s_o) \geqslant f(s_1) \geqslant \ldots$; so from well foundedness $f(s_k) = f(s_{k+1}) = \ldots$ for some k. $s_k \models Q_i$ for some i; but then so does every s_m, $m > k$, from (iii). So $s_o s_1 \ldots$ is not weak fair, from (v).

Conversely, abbreviate the right hand side of the fixpoint equation 4.2 as F(H); define

$$f(s) = \min \{\lambda \ / \ s \models F^\lambda\}$$

and take $s \models Q_i$ iff

$$s \models \bigwedge_j \neg B_j \vee EA(\mathrm{WDO}, B_i \wedge wp(C_i, \bigvee_{\lambda < f(s)} F^\lambda), \bigvee_{\lambda < f(s)} F^\lambda))$$

i.e. iff s terminates, or satisfies the i^{th} component of

$$F(\bigvee_{\lambda < f(s)} F^\lambda)$$

Then (i) - (v) follow, using Lemma 4.1 and the definition of EA.

Example: Consider the standard example

C : wdo $b \rightarrow$ skip $\square$ $b \rightarrow b :=$ false od

We want to check that $wp(C, \underline{true}) \equiv \mathcal{T}(C) \equiv \underline{true}$

Writing $G(C1,R) \equiv EA(C, b \wedge \boxed{C1}\ R, R)$, for any C1,

$\mathcal{T}(C)$ is the strongest solution to

$$X \equiv F(X)$$

where $F(X) \equiv \neg b \vee G(\underline{skip}, X) \vee G(b := \underline{false}, X)$

The iteration of F^{λ} goes:

$$F^{o} \equiv \underline{false}$$

$$F^{1} \equiv F(\underline{false})$$

$$\equiv \neg b, \text{ since } G(C, \underline{false}) \equiv \underline{false}, \text{ all } C.$$

$$F^{2} \equiv \neg b \vee G(\underline{skip}, \neg b) \vee G(b := \underline{false}, \neg b)$$

Since

$$b \wedge \boxed{\underline{skip}}\ \neg b \equiv b \wedge \neg b \equiv \underline{false}$$

$$G(\underline{skip}, \neg b) \equiv \underline{false}$$

While $G(b := false, \neg b) \equiv EA(C, b \wedge \boxed{b:=\underline{false}}\ \neg b, \neg b)$

$$\equiv EA(C, b, \neg b)$$

$$\equiv b$$

So $F^{2} \equiv \neg b \vee b \equiv \underline{true};$

and $F^{\lambda} \equiv F^{2} \equiv \underline{true}, \lambda \geqslant 2$

Finally, for the predicates Q_i of 4.3

$$Q_1 \equiv \neg b$$

$$Q_2 \equiv \neg b \vee b \equiv \underline{true}$$

5. Conclusions

The analogues for SDO of the results in Section 4 appear to be rather more complex than for WDO, as will be clear from the Lehmann, Pnueli, Stavi investigation of (strong) fairness. This fact, with the pragmatic considerations discussed in Section 3, heightens the author's concern that weak rather than strong fairness is the appropriate constraint to use. Further theoretical work is needed, however, to back up this intuition - which so far rests on purely negative evidence.

References:

1. Apt, K.R, E.-R, Olderog: Proof rules dealing with fairness. Bericht Nr. 8104, Institut fur Informatik u. Praktische Mathematik, Kiel University (1981)

2. Apt, K.R., & G.D. Plotkin: A Cook's Tour of countable nondeterminism. Technical Report, Department of Computer Science, University of Edinburgh, (1980) - (to appear in Proc. I.C.A.L.P. 81, Springer-Verlag (1981))

3. Boom, H.J.: A weaker precondition for loops. Preprint. Mathematisch Centrum, Amsterdam (1978)

4. Dijkstra, E.W.: A Discipline of Programming, Prentice-Hall (1976)

5. Hithcock, P. & D. Park: Induction rules and termination proofs. pp. 225-251 in: Automata, Languages and Programming, ed. Nivat, North Holland (1973)

6. Lehmann, D., A. Pnueli & J. Stavi: Impartiality, justice and fairness : the ethics of concurrent termination, to appear in Proc. I.C.A.L.P.81, Springer-Verlag (1981)

7. Owicki, S. & L. Lamport: Proving liveness properties of concurrent programs. Technical Report. SR1 International, Stanford (1980)

8. Park, D.: Fixpoint induction and proofs of program properties; pp. 59-78 in: Machine Intelligence 5, ed. Meltzer & Michie, Edinburgh University Press (1969)

9. Park, D.: On the semantics of fair parallelism. pp. 504-526 in: Abstract Software Specificttions. Springer Lecture Notes in Computer Science 86 (1980)

10. Park, D.: Concurrency and automata on infinite sequences in: Proc. GI Conference on Theoretical Computer Science, Karlsruhe, March 1981. Springer Lecture Notes in Computer Science (1981)

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