TITLE:
A Predicate Transformer for Weak Fair Iteration (Mathematical Studies of Information Processing)

AUTHOR(S):
PARK, DAVID

CITATION:
PARK, DAVID. A Predicate Transformer for Weak Fair Iteration (Mathematical Studies of Information Processing). 数理解析研究所講究録 1982, 454: 211-228

ISSUE DATE:
1982-04

URL:
http://hdl.handle.net/2433/103001

RIGHT:
A Predicate Transformer for Weak Fair Iteration

By

David Park


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, \(\text{wdo-}\) and \(\text{sdo-}\) statements, are added to the language of guarded commands as varieties of \(\text{do-}\) 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:

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

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

\[\text{do } B_1 + C_1 \parallel B_2 + C_2 \parallel \ldots \parallel B_n + C_n \text{ od}\]

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

\[\text{do } b + \text{skip} \parallel b + b := \text{false end} \text{ od}\]

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 \(T(\text{WDO}), T(\text{SDO})\) of Section 3 are simplifications of transformations first described by them (I was previously in doubt whether any such \(T(\text{SDO})\) 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 μ-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 \texttt{wdo}, \texttt{sdo} for \texttt{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 \(A \land B\) rather than \(A \cap B\).

More problematically, we will use infinite combinations such as \(\bigvee_{\lambda<\alpha} P_\lambda\), for transfinite \(\alpha\) and for an indexed set \(\{P_\lambda\}\) 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 \(\square, \Diamond\) 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

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

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

\[ s \models \square R \text{ iff for all } s', <s, s'> \in M(C), s' \not\models R \]
\[ s \models \Diamond R \text{ iff there exists } s', <s, s'> \in M(C) \text{ and } s' \not\models R \]
\[ \wp(C, R) \equiv T(C) \land \square R \]

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

\[ \wp(C, \text{true}) \equiv T(C) \]

since \[ \square \text{true} \equiv \text{true} \] 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 \( \square R \).

Fixpoints: to obtain a \( \wp(\text{DO}, R) \) which is correct when there is unbounded nondeterminism, and for our \( \wp(\text{DWO}, R) \) 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 \( s \models (X \Rightarrow Y), \forall s \)) 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 \( \mu F \)) and a weakest solution (the maximal fixpoint \( \nu F \)).

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

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:

$$b := \text{true}; \ z := 0;$$

$$\text{wdo} \ b + z := z+1 \quad \square \quad b + b := \text{false} \quad \text{od}$$

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)

$$b := \text{false}; \ z := ? ;$$

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

2.1 Iterations and Unbounded Nondeterminism

There is a definitional problem, illustrated by the example

$$C : \text{do} \ z < 0 \rightarrow z := ? \quad \square \quad z > 0 \rightarrow z := z-1 \quad \text{od}$$

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

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

where $H_i$ is defined inductively by

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

$$H_{i+1} \equiv H_i \lor ((z < 0 \Rightarrow wp(z := ?, H_i)) \land (z > 0 \Rightarrow wp(z := z-1, H_i)))$$

4
which solves as
\[ H_i \equiv (0 \leq z \leq i) \]
noting that
\[
\begin{align*}
wp(z := z, 0 \leq z \leq i) & \equiv false \\
wp(z := z-1, 0 \leq z \leq i) & \equiv 0 < z \leq i+1
\end{align*}
\]

But then
\[ T(C) \equiv \bigvee H_i \equiv (z > 0) \]

This is anomalous. We expect \( T(C) \equiv true \).

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

\[
T(C) \equiv H \text{ where } H \text{ is the strongest solution to }
\begin{align*}
H & \equiv (z < 0 \Rightarrow wp(z := ?, H)) \\
& \wedge (z > 0 \Rightarrow wp(z := z-1, H))
\end{align*}
\]

(* *) now determines \( T(C) \equiv true; \ H \equiv true \) 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

\[
do \ldots \; \Box B_i \rightarrow C_i \Box \; \ldots \; od
\]

Replace the \( wp(\text{DO}, R) \) definition by:

\[
2.0.1. \ wp(\text{DO}, R) \text{ is the strongest solution } H \text{ to }
H \equiv (\bigwedge_i \neg B_i \Rightarrow R) \wedge \bigwedge_i (B_i \Rightarrow wp(C_i, H))
\]

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

\[
\begin{align*}
(\text{i}) & \; F(wp(\text{DO}, R)) \Rightarrow wp(\text{DO}, R); \text{ so } \mu F \Rightarrow wp(\text{DO}, R) \\
& \text{ by Fixpoint Induction} \\
(\text{ii}) & \; \text{If } H \equiv F(H) \text{ and } s \not\models \neg H \\
& \text{ then } s \not\models \neg wp(\text{DO}, R); \\
& \text{ so } wp(\text{DO}, R) \Rightarrow \mu F.
\end{align*}
\]

But we omit the details of this scheme. We should note the case
when \( R \equiv true \) and each \( C_i \) terminates:
2.0.2. If \( B_i \Rightarrow T(C_i) \) then \( T(\text{DO}) \) is the strongest solution to
\[
H \equiv \bigwedge_i (B_i \Rightarrow C_i, H)
\]

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
\[
\wp(x := E, R) \equiv \bigwedge_{E \in \text{d}(D)} \wp(x := E', R)
\]
where \( \text{d}(E) \) 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
\[
F(\bigvee_{i=0}^\infty H_i) \neq \bigvee_{i=0}^\infty F(H_i)
\]
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 \( F(H_i) \equiv \text{false} \) for all \( i \); but \( \bigvee H_i \equiv \text{true} \).
So
\[
F(\bigvee H_i) \equiv \wp(x := ?, \text{true}) \equiv \text{true} \neq F(H_i)
\]
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

\[
\text{do } z:=0 + z:=z+1; \ b:=\neg b \ \text{if } z>0 + z:=z-1; \ \text{if } b \rightarrow \text{write (0)} \ \text{if } b \rightarrow \text{write (1)} \ \text{fi od}
\]

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.

**Definition:** A statement Cl is a slice of statement C2 iff

\[wp(C2,R) \Rightarrow wp(C1,R)\]

for all predicates R.

Thus, \(x := 1\) is a slice of \(x := ?\). If we admit as "implementations" of C2 any implementation of a slice Cl, the perplexity disappears (presumably Cl is "minimal" - i.e. a deterministic slice). The slice Cl 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 \((E_i = T(C_{i}))\) then \(s \not\models T(DO)\) iff there exists a well-ordering \((W,\succ)\) and a partial map \(f : s \rightarrow W\) with

(i) \(f(s')\) defined, \(<s', s'> \in M(C)\)

imply \(f(s'')\) defined, \(f(s') \succ f(s'')\)

(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 \( F^\lambda \) for ordinals \( \lambda \), by

\[
F^0 = \text{false} \\
F^\lambda = F\left( \bigvee_{\lambda < \alpha} F^\alpha \right)
\]

**Theorem 2.4.2:** The strongest solution to

\[ X \equiv F(X) \]

\( F \) monotone, is \( F^\alpha \), some ordinal \( \alpha \).

[ 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 \( W = \alpha \) 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 \( \omega^\omega \) may be necessary (see [1]).

3. **Strong and weak fairness**

"Fairness" is to be expected when

\[ \text{DO : do ... \text{ break } B_i \Rightarrow C_i \text{ ... od} } \]

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 \( \langle s,s' \rangle \in M(Q) \) \( \notin s \models B_i \)

**Definitions:** A finite or infinite sequence \( s_0 s_1 \ldots \) 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 **weak fair** if, for each \( j \),

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

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

The sequence is **strong fair** 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 \( do \) by \( wdo \); or by \( sdo \) if only strong fair sequences are considered.

The effect of replacing \( do \) by \( wdo \), or \( wdo \) by \( sdo \) is to increase \( T(Do) \), without affecting the relation \( M(Do) \). As an example, the statements

\[
\begin{align*}
wdo \ b & + \text{skip} \ b + b := \text{false} \ od \\
\text{sdo} \ b2 + b1 & := \neg b1 \\
\square b1 + b2 & := \text{false}; b1 := \text{false} \ od
\end{align*}
\]

both have guaranteed termination, though neither need terminate with weaker constraints. In the first case, infinite repetition of the \text{skip} is not weak fair, since the other guard would remain continuously true. In the second case, infinite repetition of \( b1 := \neg b1 \) 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 \( \text{wdo} \) or \( \text{sdo} \). Apt & Olderog [1] prove a converse result, that both varieties of fair iteration can be simulated, using statements involving "? " and ordinary \( do \)-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 \( \text{wdo} \) justification suffers from a technical complication, we consider the \( \text{sdo} \) transformation first.
3.1.1. strong fairness

\[ \text{SDO : sdo} \quad \Box B_i \rightarrow C_i \; \Box \ldots \; \text{od} \]

\[ \text{T(SDO) : } \ldots \; z_i := ? \; \ldots \; \]

\[ \text{do} \quad \ldots \]

\[ \Box B_i \land \bigwedge_j (B_j \rightarrow z_i \leq z_j) \rightarrow C_i; \; z_i := z_i + 1 + ? \]

\[ \text{od} \quad \ldots \]

To see that SDO, T(SDO) are equivalent (disregarding the introduction of \( z_i \)), consider first any finite or strong fair infinite SDO-sequence \( s_0 s_1 \ldots \). 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) \( \min \{ k / k \geq j, C_k : s_k \leftarrow s_{k+1} \} \)

or (ii) if there is no such \( k \), some \( k \) such that

\[ m \succ k \Rightarrow s_m \models \neg B_i \quad \text{for all } m \]

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

\[ \text{WDO : wdo} \quad \Box B_i \rightarrow C_i \; \Box \ldots \; \text{od} \]

\[ \text{T(WDO) : } \ldots \; z_i := ? \; \ldots \]

\[ \text{do} \quad \ldots \]

\[ \Box \bigwedge_j (z_i \leq z_j) \rightarrow \text{if } B_i \rightarrow C_i \; \Box \neg B_i \rightarrow \text{skip fi;} \]

\[ z_i := z_i + 1 + ? \]

\[ \text{od} \quad \ldots \]

\( \sqrt{B_j} \land \)
Proceeding as in 3.1.1., the complication arises in simulating a particular WDO-sequence $s_0, s_1, \ldots$. 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 $k = \min \{ m/C_i : s_m \rightarrow s_{m+1}, m \geq j \}$

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:

\[
T_w(SDO) : \ldots ; b_i := \text{false}; \ldots
\]

\[
\ldots ; z_i := 0 ; \ldots
\]

\[
\text{wdo} \ldots
to
\]

\[
\Box \bigwedge_j \left( (B_j \land b_j) \Rightarrow z_i \leq z_j \bigvee \right) B_j
\]

\[
+ \text{ if } B_i \land C_i; z_i := z_i + 1; b_i := \text{false}
\]

\[
\Box \neg B_i \land b_i := \text{true fi}
\]

\[
\text{od}
\]

Justification: An SDO-sequence is simulated by the T_w(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 $T_w(SDO)$ 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, $T_w(SDO)$ 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:

\[
\begin{align*}
  z &:= 1 \\
  \text{do} & \quad \ldots \ldots \\
  \text{if } B_i & \quad z := i + 1 \\
  \text{skip } & \quad z := 1 \\
  \text{od}
\end{align*}
\]

3.3.2. strong fairness

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

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

\[
\begin{align*}
  z_i &:= z_i + ? + 1 \text{ by } z_i := \max_{j} (z_j) + 1.
\end{align*}
\]

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

\[ X \equiv A \land \bigwedge_{i} (B_i \Rightarrow wp(C_i, X \lor Y)) \]

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 \( i \geq 0 \) \land \( (B_j \Rightarrow \neg(C_j) \) for all \( j > 0 \).

Proof: \( \Rightarrow \) Let \( H \equiv EA(WDO, A, Y) \) and let \( s_{o}s_{1} \ldots \) be a WDO-sequence with \( s_{1} \models \neg Y, i > 0 \), \( s_{o} \models H \). If \( s_{1} \models H \) then \( s_{1} \models B_{j} \Rightarrow wp(C_{j}, H \lor Y) \)

for each \( j \), from the fixpoint equation for EA(WDO, A, Y). So \( s_{i+1} \models H \lor Y \); 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.

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

\[ \hat{H} \Rightarrow A \land \bigwedge_{i} (B_{i} \Rightarrow wp(C_{i}, \hat{H} \lor Y)) \].

Clearly, \( \hat{H} \Rightarrow A \). Suppose

\[ s = s_{o} \models \hat{H} \land A \land B_{i} \land \neg wp(C_{i}, \hat{H} \lor Y) \].

Then there exists \( s_{1} \) such that \( C_{1} : s_{o} \mapsto s_{1} \), and \( s_{1} \models \neg(\hat{H} \lor Y) \). Since \( s_{1} \models \neg \hat{H} \), there is a WDO-sequence \( s_{1}s_{2} \ldots s_{n} \) with \( s_{i} \models \neg Y, i > 0 \) and \( s_{n} \models \neg A \). But then \( s_{o}s_{1} \ldots s_{n} \) contradicts \( s_{o} \models \hat{H} \).
Corollary: \( \text{EA}(\text{WDO}, A, Y) \) is monotone in \( A, Y \). [Since the equivalent predicate is clearly monotone in \( A, Y \).] The required \( \text{wp} \) is now

\[
\text{wp}(\text{WDO}, R) \text{ is the strongest solution } H \]
\[
H \equiv (\bigwedge_i \neg B_i \land R) \lor \bigvee_i \text{EA}(\text{WDO}, B_i \land \text{wp}(C_i, H), H)
\]

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

**Justification of 4.2:**

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

\[
\Rightarrow \quad \text{Use standard Fixpoint Induction; we must show} \quad (\bigwedge_i \neg B_i \land R) \lor \bigvee_i \text{EA}(\text{WDO}, B_i \land \text{wp}(C_i, \hat{H}), H) = \hat{H}
\]

Clearly \( (\bigwedge_i \neg B_i \land R) = \hat{H} \); suppose

\[
s_o \models \text{EA}(\text{WDO}, B_i \land \text{wp}(C_i, \hat{H}), H)
\]

From the Lemma, if a WDO-sequence avoids \( \hat{H} \), it passes only through states \( s_i \models B_i \land \text{wp}(C_i, \hat{H}) \). 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 \( s_o \not\models \hat{H} \), since termination with \( R \) is guaranteed.

\[
\Rightarrow \quad \text{suppose } s_o \models \neg H; \text{ we construct an embarrassing WDO-sequence } s_o \ldots s_1 \ldots s_2 \ldots \text{ with each } s_i \models \neg H, \text{ by induction on } i. \text{ Suppose } s_i \models \neg H, \text{ then }
\]

\[
s_i \models \neg \text{EA}(\text{WDO}, B_j \land \text{wp}(C_j, H), H)
\]

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

\[
s' \models \neg H \text{ with } s' \models \neg (B_j \land \text{wp}(C_j, H)).
\]

If \( s' \models \neg B_j \), take \( s_{i+1} = s' \); otherwise choose \( C_j : s' \models s_{i+1} \) 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 \( s_0, s_1, \ldots \)

or (ii) some \( s_i \Vdash \bigwedge_i \neg B_i \land \neg R \), and there is a finite WDO-sequence reaching \( \neg R \).

Finally, we can obtain our analogue of the Lehnmann, Pnueli & Stavi result \([6]\).

**Corollary 4.3:**

If \( B_i \Rightarrow T(C_i) \), then \( s \models T(WDO) \) iff there exist a well ordering \((W, \succ)\), a partial map \( f : s \rightarrow W \), and predicates \( Q, Q_i \) with

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

(ii) if \( s \models Q, C_i : s \rightarrow s' \) then \( s' \Vdash Q \) and \( f(s) \succ f(s') \)

(iii) if \( s \models Q_i, C_j : s \rightarrow s' \), \( f(s) = f(s') \) then \( s' \Vdash Q_i \)

(iv) if \( s \models Q_i, C_i : s \rightarrow s' \) then \( f(s) \succ f(s') \)

(v) \( Q_i \Rightarrow B_i \lor \bigwedge_j \neg B_j \)

**Proof:** analogous to the argument in 2.4. Suppose (i) - (v) are satisfied for \( s \), and \( s_0, s_1, \ldots \) is an infinite WDO-sequence from \( s = s_0 \). From (ii) every \( f(s_i) \) is defined, and \( f(s_0) \succ f(s_1) \succ \ldots \); so from well foundedness \( f(s_k) = f(s_{k+1}) = \ldots \) for some \( k \). \( s_k \Vdash Q_i \) for some \( i \); but then so does every \( s_m, m > k \), from (iii). So \( s, s_1, \ldots \) is not weak fair, from (v).

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

\[
    f(s) = \min \{ \lambda \mid s \models F^\lambda \}
\]

and take \( s \models Q_i \) iff

\[
    s \Vdash \bigwedge_j \bigvee_i (\neg B_j \lor \text{EA}(WDO, B_i \land \text{wp}(C_i, \bigvee_{\lambda < f(s)} F^\lambda), \bigvee_{\lambda < f(s)} F^\lambda))
\]

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

\[ C : \text{wdo } b \rightarrow \text{skip } [] b \rightarrow b : = \text{false } \text{od} \]

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

Writing \( G(C_l, R) \equiv \text{EA}(C_l, b \land [C_l] R, R) \), for any \( C_l \),

\( T(C) \) is the strongest solution to

\[ X \equiv \text{F}(X) \]

where \( \text{F}(X) \equiv \neg b \lor G(\text{skip}, X) \lor G(b : = \text{false}, X) \)

The iteration of \( F^\lambda \) goes:

\[ F^0 \equiv \text{false} \]

\[ F^1 \equiv F(\text{false}) \]

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

\[ F^2 \equiv \neg b \lor G(\text{skip}, \neg b) \lor G(b : = \text{false}, \neg b) \]

Since

\[ b \land [\text{skip}] \neg b \equiv b \land \neg b \equiv \text{false} \]

\[ G(\text{skip}, \neg b) \equiv \text{false} \]

While

\[ G(b : = \text{false}, \neg b) \equiv \text{EA}(C, b \land [b : = \text{false}] \neg b, \neg b) \]

\[ \equiv \text{EA}(C, b, \neg b) \]

\[ \equiv b \]

So

\[ F^2 \equiv \neg b \lor b \equiv \text{true}; \]

and

\[ F^\lambda \equiv F^2 \equiv \text{true}, \ \lambda \geq 2 \]

Finally, for the predicates \( Q_1 \) of 4.3

\[ Q_1 \equiv \neg b \]

\[ Q_2 \equiv \neg b \lor b \equiv \text{true} \]

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:

   Bericht Nr. 8104, Institut fur Informatik u. Praktische Mathematik, 
   Kiel University (1981)

   Technical Report, Department of Computer Science, University of Edinburgh, 

   Centrum, Amsterdam (1978)


   pp. 225-251 in: Automata, Languages and Programming, ed. Nivat, 
   North Holland (1973)

   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 

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

   Abstract Software Specifications. Springer Lecture Notes in Computer 
   Science 80 (1980)

    Springer Lecture Notes in Computer Science (1981)

    to Programming Language Theory. MIT Press (1977)