

Temporal Predicate Transformers and Fair Termination

Joseph M. Morris

Department of Computing Science
University of Glasgow
Glasgow G12 8QQ
Scotland, UK



Temporal Predicate Transformers and Fair Termination, Joseph M. Morris, Acta Informatica 27, 287-313 (1990). © Springer-Verlag 1990

Summary. It is usually assumed that implementations of nondeterministic programs may resolve the nondeterminacy arbitrarily. In some circumstances, however, we may wish to assume that the implementation is in some sense fair, by which we mean that in its long-term behaviour it does not show undue bias in forever favouring some nondeterministic choices over others. Under the assumption of fairness many otherwise failing programs become terminating. We construct various predicate transformer semantics of such fairly-terminating programs. The approach is based on formulating the familiar temporal operators *always*, *eventually*, and *infinitely often* as predicate transformers. We use these operators to construct a framework that accommodates many kinds of fairness, including varieties of so-called weak and strong fairness in both their all-levels and top-level forms. Our formalization of the notion of fairness does not exploit the syntactic shape of programs, and allows the familiar nondeterminacy and fair nondeterminacy to be arbitrarily combined in the one program. Invariance theorems for reasoning about fairly terminating programs are proved. The semantics admits probabilistic implementations provided that unbounded fairness is excluded.

1 Introduction

It is a convenient abstraction to assume that implementations of nondeterministic programs have components called "arbiters" that resolve the nondeterminacy. In some circumstances we may wish to assume that the arbiter is in some sense "fair", by which we mean that in its long-term behaviour it does not show undue bias in favouring some choices over others. Under the assumption of fairness many otherwise failing programs become terminating. Consider the recursive procedure

$g0: \text{skip} \square g0$

where \square is the nondeterministic choice operator. The usual implementation is assumed to employ an unpredictable arbiter in which case $g0$ is not guaranteed to terminate, and indeed we may calculate $\text{wp}(g0, \text{true}) = \text{false}$. Weakest preconditions are in this sense conservative, equating with **abort** every program that may fail to terminate. However, if we may assume that the arbiter behaves fairly to the extent that it will not forever ignore the first of the two statements in $g0$ then $g0$ terminates; we would like to have a formal system in which we could prove this. In short, we would like to construct a predicate transformer semantics of programs when the intended implementation is fair. Among the questions to be answered are: 'What is "fair"?' and 'What distinctive properties, if any, do fairly terminating programs have?'.

Initially we will not be concerned with fairness but with properties of programs as predicate transformers, and with representing temporal operators such as *always* and *eventually* as predicate transformers. Subsequently we will use these temporal operators to define various notions of fairness. We will consider how the presence of fairness affects the properties of programs, formulate some termination

theorems, and discuss implementation issues. Although we shall be introducing various definitions of fairness, this is not the sole point of the work. The main point is the attempt to introduce a theoretical vocabulary and style that can accommodate different notions of fairness, and that admits of effective reasoning about fairness at the programming level.

2 Predicate Transformers

We assume for the purposes of the paper that programs operate on a fixed set of variables, whose set of possible values defines the state space of the program. Assertions are predicates in which program variables may occur free; assertion X universally quantified over the program variables is denoted by $[X]$. Relation \leq on assertions defined by $X \leq Y = [X \Rightarrow Y]$ is easily shown to be a partial ordering. We will use capital letters near the end of the alphabet to stand for arbitrary assertions. T and F are the assertions satisfied by all states and by no state, respectively. We will use the following operators, grouped in order of decreasing precedence: $.$ (function application); \neg ; \wedge , \vee ; \Rightarrow ; \equiv . We give programs a semantics by regarding them as functions on assertions, in the usual way [2]. Function f on assertions is said to be conjunctive iff

$$[f.X \wedge f.Y \equiv f.(X \wedge Y)]$$

for all X, Y ; it is "monotonic" iff

$$[X \Rightarrow Y] \Rightarrow [f.X \Rightarrow f.Y]$$

for all X, Y . Function f on a pair of assertions is "left conjunctive" iff

$$[f.(X,Z) \wedge f.(Y,Z) \equiv f.(X \wedge Y, Z)]$$

for all X, Y, Z ; less precisely, we may just say $f.(X,Y)$ is "conjunctive in X ". Similar remarks hold for "right conjunctivity", "left monotonicity", and "right monotonicity". Furthermore we say that f is "conjunctive" iff

$$[f.(W,X) \wedge f.(Y,Z) \equiv f.(W \wedge Y, X \wedge Z)]$$

for all W, X, Y, Z . Conjunctivity is stronger than the combination of left and right conjunctivity. Monotonicity of $f.(X,Y)$ is defined similarly; it is equivalent to the combination of left and right monotonicity. Unless the contrary is explicitly indicated all functions are from assertions, or pairs of assertions, to assertions, and monotonicity etc. is with respect to the \leq ordering.

Lemma 1 *Conjunctive functions are monotonic.*

Proof. (See the note on proofs below.) For any conjunctive function f

$$\begin{aligned} & [X \Rightarrow Y] \Rightarrow [f.X \Rightarrow f.Y] \\ =\{ \text{detach } [X \Rightarrow Y] \} & \\ & [f.X \Rightarrow f.Y] \\ =\{ \text{calculus} \} & \\ & [f.X \equiv f.X \wedge f.Y] \\ =\{ f \text{ conjunctive} \} & \\ & [f.X \equiv f.(X \wedge Y)] \\ \Leftarrow \{ \text{calculus} \} & \\ & [X \equiv X \wedge Y] \\ =\{ \text{calculus} \} & \\ & [X \Rightarrow Y] \\ =\{ \text{detached axiom} \} & \\ & \text{true} \quad (\text{End}) \end{aligned}$$

Note on proofs: In proofs, the justification of each proof step is written between chain brackets. The frequently occurring hint "calculus" records an appeal to familiar mathematics, in particular predicate calculus, arithmetic, or set theory. Proofs usually proceed by reducing the demonstrandum to 'true', some using \Leftarrow (pronounced "follows from"): $A \Leftarrow B$ is the same as $B \Rightarrow A$. The steps often appeal to the rule

$$[A \Rightarrow D] \Leftarrow [A \Rightarrow B] \wedge [B \Rightarrow D].$$

For example,

$$\begin{aligned} & [A \Rightarrow D] \\ \Leftarrow \{ \text{hint why } [A \Rightarrow B] \} & \\ & [B \Rightarrow D] \\ \Leftarrow \{ \text{hint why } [C \Rightarrow D] \} & \\ & [B \Rightarrow C] \\ = \{ \text{hint why } [B \Rightarrow C] \} & \\ & \text{true.} \end{aligned}$$

An application of the deduction theorem of predicate calculus is indicated by

$$\begin{aligned} & [B] \Rightarrow [C] \\ = \{ \text{detach } [B] \} & \\ & [C] \end{aligned}$$

– the subsequent proof of $[C]$ may appeal to $[B]$. Inference by universalisation is recorded as

$$\begin{aligned} & (\forall x: Q) \\ \Leftarrow \{ \text{universalisation} \} & \\ & Q \qquad \qquad \qquad (\text{End}) \end{aligned}$$

Formally, we regard every program s as a function on assertions that maps an arbitrary postcondition R to the weakest precondition such that s establishes R . It is customary to denote the weakest precondition such that s establishes R by $wp(s,R)$; however, we will prefer the more economic notation $s.R$. We define the semantics of straight-line programs thus (explanations follow):

$$\begin{aligned} s.R = & \\ & \bullet s \text{ is } \mathbf{skip}: R \\ & \bullet s \text{ is } x := e: R(x/e) \\ & \bullet s \text{ is } u;v: u.(v.R) \\ & \bullet s \text{ is } \mathbf{if } b \mathbf{ then } u \mathbf{ else } v \mathbf{ fi}: (b \Rightarrow u.R) \wedge (\neg b \Rightarrow v.R) \\ & \bullet s \text{ is } u \square v: u.R \wedge v.R \end{aligned}$$

$R(x/e)$ stands for R with each occurrence of variable x in R replaced by expression e , and b stands (here and in the sequel) for a boolean expression. Choice and guarding usually appear in combination as the familiar **if...fi** statement, but they are here separated because we want to put choice under the microscope. The theory we will develop does not at all depend on this separation, which is introduced solely for clarity of the exposition. We give the semicolon a higher operator precedence than choice. We are rather lax in our requirements on a fair arbiter: we ask it to exhibit fairness only in computations of unbounded length, and this requires recursion or looping which we'll be considering later. Except where otherwise indicated small letters near the end of the alphabet will denote arbitrary programs.

We shall have need of constant predicate transformers. It will be convenient to double up on notation and denote by P the constant predicate transformer whose value is predicate P , i.e.

$$[P.X \equiv P] \text{ for all } X.$$

Property 1 *Programs are conjunctive, i.e. $[s.X \wedge s.Y \equiv s.(X \wedge Y)]$.*

Proof. The proof – at this stage just for straight-line programs – is by routine induction on the structure of s and is omitted; see the proof of Property 3 for an example of the style. (End)

Property 2 *Programs are monotonic.*

Proof. Lemma 1 and Property 1. (End)

We denote by $g:G$ the introduction of a procedure with name g and body G ; we assume that all recursion is tail-recursion and that there is no mutual recursion. The letter g will always stand for a procedure. For any program s , program (or predicate transformer) t , and component statement u of s we denote by $s(u/t)$ the program got by replacing each occurrence of statement u in s with t (it is routine to define this substitution mechanism formally by structural induction, but that hardly seems necessary). In the special case of procedure $g:G$ we shall abbreviate $G(g/t)$ by $g(t)$; we are then especially interested in the case that t is a constant predicate transformer. Operationally, $g(P).R$ is the weakest precondition such that an execution of the body of g leads to an immediately following invocation of g in a state satisfying P or else establishes R without further invoking g . We may somewhat loosely talk of $g(P).R$ as a function of the pair (P,R) .

Example 1 Consider **DO: if b then s; DO else skip fi.**

$$\begin{aligned}
& \text{DO}(P).T \\
= & \{\text{definition of DO}(P)\} \\
& \mathbf{if\ b\ then\ } s; P \mathbf{ else\ skip\ fi.} T \\
= & \{\text{semantics}\} \\
& (b \Rightarrow (s; P).T) \wedge (\neg b \Rightarrow \mathbf{skip}.T) \\
= & \{\text{semantics}\} \\
& (b \Rightarrow s.P) \wedge (\neg b \Rightarrow T) \\
= & \{\text{calculus}\} \\
& \neg b \vee s.P \qquad \qquad \qquad \text{(End)}
\end{aligned}$$

Property 3 *$g(X).Y$ is conjunctive.*

Proof. Letting G denote the body of g we prove by structural induction

$$[G(g/X).Y \wedge G(g/P).Q \equiv G(g/X \wedge P).(Y \wedge Q)]$$

- (i) G is assignment or **skip**: reduces to Property 1.
- (ii) G is g :
$$\begin{aligned}
& G(g/X).Y \wedge G(g/P).Q \\
= & \{G \text{ is } g\} \\
& X.Y \wedge P.Q \\
= & \{\text{definitions}\} \\
& X \wedge P \\
= & \{\text{definition}\} \\
& (X \wedge P).(Y \wedge Q) \\
= & \{G \text{ is } g\} \\
& G(g/X \wedge P).(Y \wedge Q)
\end{aligned}$$
- (iii) G is h where h is a procedure different from g : reduces to Property 1 by the absence of mutual recursion.
- (iv) G is $u; v$: Assume the inductive hypothesis holds for v .
$$\begin{aligned}
& G(g/X).Y \wedge G(g/P).Q \\
= & \{G \text{ is } u; v\} \\
& ((u;v)(g/X)).Y \wedge ((u;v)(g/P)).Q \\
= & \{g \text{ does not occur in } u \text{ by the requirement of tail recursion}\} \\
& (u; v(g/X)).Y \wedge (u; v(g/P)).Q
\end{aligned}$$

={semantics of semicolon}
 $u.(v(g/X).Y) \wedge u.(v(g/P).Q)$
 ={Property 1 applied to u}
 $u.(v(g/X).Y \wedge v(g/P).Q)$
 ={inductive hypothesis for v}
 $u.(v(g/X \wedge P).(Y \wedge Q))$
 ={semantics of semicolon; g does not occur in u}
 $((u;v)(g/X \wedge P).(Y \wedge Q))$
 ={G is u;v}
 $G(g/X \wedge P).(Y \wedge Q)$
 (Tail recursion is not crucial in the above.)
 (v) G is **if ... fi**: exercise.
 (vi) G is u [] v : similar to case (iv) (End)

Property 4 $g(X).Y$ is monotonic.

Proof. Lemma 1 and Property 3. (End)

We say that predicate P is an "invariant (of g) for R", for any predicate R, iff $[P \Rightarrow g(P).R]$; P is simply an "invariant" if it's an invariant for T. Operationally P is an invariant for R iff each execution of g in a state satisfying P leads to all recursive invocations of g taking place in a state satisfying P, and either g iterates forever or establishes R. (The "establishes R" clause of the preceding sentence appeals to the tail recursiveness of g.)

Example 2 With DO as defined in Example 1 we have

P an invariant of DO
 ={definition}
 $[P \Rightarrow DO(P).T]$
 ={Example 1}
 $[P \Rightarrow \neg b \vee s.P]$
 ={calculus}
 $[P \wedge b \Rightarrow s.P]$

This is as we would like: the final line is the usual definition of invariant as applied to **do b \rightarrow s od**.

(End)

3 Temporal Predicate Transformers

For g a recursive procedure we want to formulate the weakest precondition such that a given predicate *always* holds on the initial and on each recursive invocation of g, and similarly for *eventually* and *infinitely often*. We call these "temporal predicate transformers". It turns out that temporal predicate transformers may be expressed as the least or greatest fixpoints of certain monotonic functions on predicates.

3.1 Fixpoints

For any fixpoint X of function f its defining property $X = f.X$ is called the "fixpoint property (of X with respect to f)". We denote by $(\eta X:f.X)$ the greatest fixpoint of f, and by $(\mu X:f.X)$ the least fixpoint. All monotonic functions on a complete lattice have least and greatest fixpoints. See [8] for the background on fixpoints. It is shown in [5] that assertions, partially ordered by \leq , are embedded in a complete lattice, and so monotonic functions on assertions possess least and greatest fixpoints. If Z denotes the greatest fixpoint of function f then the property that for all X

$$[X \Rightarrow f.X] \Rightarrow [X \Rightarrow Z]$$

is called the "greatest fixpoint property (of Z with respect to f)". The "least fixpoint property" is defined dually: denoting the least fixpoint of f by Y then for all X

$$[f.X \Rightarrow X] \Rightarrow [Y \Rightarrow X]$$

These fixpoint properties are instances of fixpoint induction; see [7].

3.2 "Always"

For procedure g and predicates P, R we define

$$\mathcal{A}_g.(P,R) = (\eta X: P \wedge g(X).R)$$

This fixpoint exists because $P \wedge g(X).R$ is monotonic in X for all P, R; this is easily inferred from Property 4 and predicate calculus. Operationally $\mathcal{A}_g.(P,R)$ is the weakest precondition such that P holds on each invocation of g – initial and recursive – and such that g iterates forever or establishes R. However, we will not be appealing to this operational interpretation. For brevity we will omit the subscript in $\mathcal{A}_g.(P,R)$, it being always understood to be g.

Example 3 $\mathcal{A}.(T,R)$ is the weakest precondition such that g iterates forever or establishes R; $\mathcal{A}.(T,F)$ is the weakest precondition such that g iterates forever. $\mathcal{A}.(T,T)$ is the weakest invariant of g (End)

Property 5 $\mathcal{A}.(P,R)$ is monotonic.

Proof.

$$\begin{aligned} & [P \Rightarrow Q] \wedge [R \Rightarrow S] \Rightarrow [\mathcal{A}.(P,R) \Rightarrow \mathcal{A}.(Q,S)] \\ =\{ \text{detach } [P \Rightarrow Q] \wedge [R \Rightarrow S] \} \\ & [\mathcal{A}.(P,R) \Rightarrow \mathcal{A}.(Q,S)] \\ =\{ \text{definition} \} \\ & [\mathcal{A}.(P,R) \Rightarrow (\eta X: Q \wedge g(X).S)] \\ \Leftarrow\{ \text{greatest fixpoint} \} \\ & [\mathcal{A}.(P,R) \Rightarrow Q \wedge g(\mathcal{A}.(P,R)).S] \\ =\{ \text{fixpoint property of } \mathcal{A}.(P,R) \} \\ & [P \wedge g(\mathcal{A}.(P,R)).R \Rightarrow Q \wedge g(\mathcal{A}.(P,R)).S] \\ \Leftarrow\{ \text{calculus} \} \\ & [P \Rightarrow Q] \wedge [g(\mathcal{A}.(P,R)).R \Rightarrow g(\mathcal{A}.(P,R)).S] \\ \Leftarrow\{ g(X).Y \text{ monotonic in } Y \} \\ & [P \Rightarrow Q] \wedge [R \Rightarrow S] \\ =\{ \text{detached axiom} \} \\ & \text{true} \end{aligned} \quad (\text{End})$$

Property 6 $\mathcal{A}.(P,R)$ is conjunctive.

Proof: We show $[\mathcal{A}.(P,R) \wedge \mathcal{A}.(Q,S) \equiv \mathcal{A}.(P \wedge Q, R \wedge S)]$. The proof from right to left follows easily from the monotonicity of \mathcal{A} and is left to the reader. For the other direction

$$\begin{aligned} & [\mathcal{A}.(P,R) \wedge \mathcal{A}.(Q,S) \Rightarrow \mathcal{A}.(P \wedge Q, R \wedge S)] \\ =\{ \text{definition of } \mathcal{A}.(P \wedge Q, R \wedge S) \} \\ & [\mathcal{A}.(P,R) \wedge \mathcal{A}.(Q,S) \Rightarrow (\eta X: P \wedge Q \wedge g(X).(R \wedge S))] \\ \Leftarrow\{ \text{greatest fixpoint property} \} \\ & [\mathcal{A}.(P,R) \wedge \mathcal{A}.(Q,S) \Rightarrow P \wedge Q \wedge g(\mathcal{A}.(P,R) \wedge \mathcal{A}.(Q,S)).(R \wedge S)] \\ =\{ \text{calculus; } g(X).Y \text{ conjunctive} \} \\ & [\mathcal{A}.(P,R) \wedge \mathcal{A}.(Q,S) \Rightarrow P \wedge Q \wedge g(\mathcal{A}.(P,R)).R \wedge g(\mathcal{A}.(Q,S)).S] \\ \Leftarrow\{ \text{calculus} \} \\ & [\mathcal{A}.(P,R) \Rightarrow P \wedge g(\mathcal{A}.(P,R)).R] \wedge [\mathcal{A}.(Q,S) \Rightarrow Q \wedge g(\mathcal{A}.(Q,S)).S] \\ =\{ \text{fixpoint properties} \} \\ & \text{true} \end{aligned} \quad (\text{End})$$

Property 7 $\mathcal{A}.(P,R)$ is an invariant for R .

Property 8 $[\mathcal{A}.(P,R) \Rightarrow P]$.

Proofs of Properties 7, 8.

$$\begin{aligned}
& (\mathcal{A}.(P,R) \text{ an invariant for } R) \wedge [\mathcal{A}.(P,R) \Rightarrow P] \\
= \{ \text{definition} \} & \\
& [\mathcal{A}.(P,R) \Rightarrow g(\mathcal{A}.(P,R)).R] \wedge [\mathcal{A}.(P,R) \Rightarrow P] \\
= \{ \text{calculus} \} & \\
& [\mathcal{A}.(P,R) \Rightarrow P \wedge g(\mathcal{A}.(P,R)).R] \\
= \{ \text{fixpoint property of } \mathcal{A}.(P,R) \} & \\
& \text{true} \qquad \qquad \qquad \text{(End)}
\end{aligned}$$

Indeed $\mathcal{A}.(P,R)$ is maximal in that if X is an invariant for R and $[X \Rightarrow P]$ then $[X \Rightarrow \mathcal{A}.(P,R)]$; this follows from the greatest fixpoint property of $\mathcal{A}.(P,R)$.

Property 9 If P is an invariant for R then $[\mathcal{A}.(P,R) \equiv P]$.

Proof. Exercise. (End)

3.3 "Eventually"

(This and the following subsection may be omitted on first reading; 3.3 is a preparation for subsection 3.4 which in turn is not needed until section 9.) For procedure g and predicates P, R we define

$$\mathcal{F}_g.(P,R) = (\mu X: P \vee g(X).R)$$

This fixpoint exists because, as the reader may infer from Property 4 and some predicate calculus, $P \vee g(X).R$ is monotonic in X for all P, R . Operationally $\mathcal{F}_g.(P,R)$ is the weakest precondition such that g establishes R , or P holds at some invocation of g ; we will not appeal to this operational interpretation (and admit that it's not obvious). We will omit the subscript in $\mathcal{F}_g.(P,R)$, it being always understood to be g . To prove a conjunctivity property of \mathcal{F} we shall need the following lemma.

Lemma 2 For any conjunctive function $f.(X,Y)$ let $h.Y = (\mu X:f.(X,Y))$. Then $h.Y$ is conjunctive.

Proof. We show $[h.P \wedge h.Q \equiv h.(P \wedge Q)]$. For the proof from right to left

$$\begin{aligned}
& [h.(P \wedge Q) \Rightarrow h.P \wedge h.Q] \\
= \{ \text{definition of } h.(P \wedge Q) \} & \\
& [(\mu X:f.(X, P \wedge Q)) \Rightarrow h.P \wedge h.Q] \\
\Leftarrow \{ \text{least fixpoint property} \} & \\
& [f.(h.P \wedge h.Q, P \wedge Q) \Rightarrow h.P \wedge h.Q] \\
= \{ f \text{ is conjunctive} \} & \\
& [f.(h.P, P) \wedge f.(h.Q, Q) \Rightarrow h.P \wedge h.Q] \\
= \{ \text{fixpoint properties} \} & \\
& [h.P \wedge h.Q \Rightarrow h.P \wedge h.Q] \\
= \{ \text{calculus} \} & \\
& \text{true}
\end{aligned}$$

In the other direction:

$$\begin{aligned}
& [h.P \wedge h.Q \Rightarrow h.(P \wedge Q)] \\
= \{ \text{calculus} \} & \\
& [h.P \Rightarrow \neg h.Q \vee h.(P \wedge Q)] \\
= \{ \text{definition of } h.P \} & \\
& [(\mu X:f.(X, P)) \Rightarrow \neg h.Q \vee h.(P \wedge Q)] \\
\Leftarrow \{ \text{least fixpoint property} \} & \\
& [f.(\neg h.Q \vee h.(P \wedge Q), P) \Rightarrow \neg h.Q \vee h.(P \wedge Q)]
\end{aligned}$$

$$\begin{aligned}
& =\{\text{calculus}\} \\
& \quad [f.(\neg h.Q \vee h.(P \wedge Q), P) \wedge h.Q \Rightarrow h.(P \wedge Q)] \\
& =\{\text{fixpoint property of } h.Q\} \\
& \quad [f.(\neg h.Q \vee h.(P \wedge Q), P) \wedge f.(h.Q, Q) \Rightarrow h.(P \wedge Q)] \\
& =\{f \text{ is conjunctive; calculus}\} \\
& \quad [f.(h.Q \wedge h.(P \wedge Q), P \wedge Q) \Rightarrow h.(P \wedge Q)] \\
& =\{\text{fixpoint property of } h.(P \wedge Q)\} \\
& \quad [f.(h.Q \wedge h.(P \wedge Q), P \wedge Q) \Rightarrow f.(h.(P \wedge Q), P \wedge Q)] \\
& \Leftarrow \{f \text{ is conjunctive, hence left monotonic (Lemma 1)}\} \\
& \quad [h.Q \wedge h.(P \wedge Q) \Rightarrow h.(P \wedge Q)] \\
& =\{\text{calculus}\} \\
& \quad \text{true} \tag{End}
\end{aligned}$$

Property 10 $\mathcal{E}(P, R)$ is conjunctive in R .

Proof. Let $f.(X, R) = P \vee g(X).R$. By a routine application of predicate calculus the reader may infer that $f.(X, R)$ is conjunctive. Hence by Lemma 2 ($\mu X:f.(X, R)$), which is $\mathcal{E}(P, R)$, is conjunctive in R . (End)

Property 11 $\mathcal{E}(P, R)$ is monotonic.

Proof. Exercise. (End)

Property 12 $[P \Rightarrow \mathcal{E}(P, R)]$

Property 13 $[g(\mathcal{E}(P, R)).R \Rightarrow \mathcal{E}(P, R)]$

Proof of properties 12, 13. Similar to the proofs of Properties 7 and 8. (End)

\mathcal{E} and \mathcal{A} combine in a sort of semi-conjunctivity relationship:

Property 14 $[\mathcal{E}(P, Q) \wedge \mathcal{A}(R, S) \Rightarrow \mathcal{E}(P \wedge R, Q \wedge S)]$

Proof. Exercise. (Hint: The proof proceeds much as for the proof of Lemma 2 in one direction.) (End)

We mention in passing that the usual definition of $g.R$, without assumptions of fairness, is $g.R = \mathcal{E}(F, R)$, i.e. $g.R = (\mu X:g(X).R)$; the proof of Property 1 is completed in this case by a straight appeal to Property 10.

3.4 "Infinitely often"

For procedure g and predicates P, R we define

$$\mathcal{I}_g.(P, R) = \mathcal{A}_g.(\mathcal{E}_g.(P, T), R)$$

As usual we will omit the subscript g . The operational interpretation of $\mathcal{I}(P, R)$, which we will not appeal to formally, is that it is the weakest precondition such that g establishes R or iterates forever with P holding infinitely often on invocation of g .

Property 15 $\mathcal{I}(P, R)$ is conjunctive in R .

Property 16 $\mathcal{I}(P, R)$ is monotonic.

Property 17 $\mathcal{I}(P, R)$ is an invariant for R .

Property 18 $[\mathcal{I}(P, Q) \wedge \mathcal{A}(R, S) \Rightarrow \mathcal{I}(P \wedge R, Q \wedge S)]$

Proof of Properties 15, 16, 17, 18. The proofs are easy exercises in predicate calculus using the definition of $\mathcal{I}(P, R)$ and the properties of \mathcal{E} and \mathcal{A} already established; they are left to the reader. (End)

4 Fair Choice

We introduce the fair choice operator on statements, represented by \square_f . Semantically \square_f has the same predicate transformer semantics as \square ; it differs from \square in that it influences the behaviour of recursive procedures in which it occurs. The intention behind \square_f is that the choice should be a fair one; roughly speaking, we require that in a computation of unbounded length one of the two possible choices should not be unduly favoured over the other. Our task is to explicate this formally.

Because \square and \square_f have the same predicate transformer semantics it follows that the introduction of fair choice does not destroy Properties 2, 3, and 4, and we will show that Property 1 continues to hold. It is also evident that \square_f 's and \square 's are interchangeable as far as \mathcal{A} , \mathcal{E} , and \mathcal{I} are concerned.

We shall also have need of so-called "angelic choice" denoted by \square_a and defined by

$$(u \square_a v).R = u.R \vee v.R$$

Angelic choice is introduced as a mathematical device for constructing a semantics of fairness; it is not available to the programmer but arises in certain transformations of programs to be described later. Intuitively, $u \square_a v$ establishes postcondition R if either u or v establishes R. It is evident that angelic choice is monotonic, and thence it easily follows that Property 4 continues to hold in the presence of \square_a . Angelic choice is not conjunctive, however.

For s a statement possibly containing \square_f 's we denote by s° a copy of s with each \square_f in s replaced with \square_a . $s^\circ.R$ is the weakest precondition such that s *may* establish R provided a lucky choice is made at each \square_f . $g(X)^\circ.R$ is the weakest precondition such that execution of the body of g may, by making a favourable choice at each \square_f , give rise to an immediately following (recursive) invocation of g in a state satisfying X, or to termination in a state satisfying R without further invoking g.

Property 19 $[g(P)^\circ.R \wedge g(Q).S \Rightarrow g(P \wedge Q)^\circ.(R \wedge S)]$

(Roughly speaking, this says that if an action *may* achieve X and is *guaranteed* to achieve Y then it *may* achieve X and Y simultaneously.)

Proof. By structural induction on g; we leave it to the reader.

(End)

5 A Weak Fairness

We are now in a position to define g.R in the possible presence of fairness; in fact we shall do so in various ways the first of which we shall call "weak fairness". We define

$$g.R = (\mu X: \mathcal{A}.(g(X)^\circ.T, R))$$

That the least fixpoint exists follows easily from monotonicity considerations, in particular from Postulate 1 and Property 5. We note that this definition of g.R reduces to the standard definition (given in subsection 3.3) in the absence of fair choice; the reader may care to prove this.

We must show that the above definition reasonably captures the idea of fairness, and that g is conjunctive.

5.1 Justification

We justify the definition of g.R. Consider any execution – fair or unfair – of g in an initial state satisfying g.R; we shall show that if g fails to terminate then the implementation is – in a sense to be

made precise – unfair. First observe that consequent upon the fixpoint property of $g.R$ and Property 7 $g.R$ is an invariant for R ; therefore g establishes R or iterates forever. Moreover if g iterates forever then we can exhibit a predicate Q_0 such that eventually every invocation of g takes place in a state satisfying $g(Q_0)^\circ.T$ but never satisfying Q_0 . Recalling the operational interpretation of $g(X).Y$ from section 4, the implementation is then unfair in the sense that it has infinitely many opportunities to make good (i.e. progressive) choices at the \llbracket_f 's and so establish Q_0 or terminate, but never takes them. We proceed to exhibit Q_0 .

Firstly we summarise the theory of constructed fixpoints:

Lemma 3 For monotonic function $f.X$ define the transfinite sequence of predicates

$$\begin{aligned} Z.0 &= F \\ Z.(i+1) &= f.(Z.i) \text{ for successor ordinals } i+1 \\ Z.m &= (\exists i: i < m: Z.i) \text{ for limit ordinals } m. \end{aligned}$$

Then

- (i) $i \leq j \Rightarrow [Z.i \Rightarrow Z.j]$ for all ordinals i, j
- (ii) $(\mu X: f.X) = Z.i$ for some i .

Proof. It is proved in [5] that the complete lattice of assertions partially ordered by \leq has bottom element F and least upper bounds satisfying $\text{lub}\{P.i: i \in S\} = (\exists i: i \in S: P.i)$ where the $P.i$'s are assertions, S is any set, and lub yields least upper bounds. The proof then follows from a version of the Knaster-Tarski theorem proved in [1]. (End)

Now define the transfinite sequence of predicates

$$\begin{aligned} Z.0 &= F \\ Z.(i+1) &= \mathcal{A}.(g(Z.i)^\circ.T, R) \text{ for successor ordinals } i+1 \\ Z.m &= (\exists i: i < m: Z.i) \text{ for limit ordinals } m. \end{aligned}$$

Lemma 4 $g.R = Z.j$ for some j .

Proof. Lemma 3(ii) with $\mathcal{A}.(g(X)^\circ.T, R)$ for $f.X$ (End)

Lemma 5 $[Z.j \Rightarrow g(Z.j).R]$ for all $j \geq 0$.

Proof. By transfinite induction on j .

- (i) $j = 0$:
 - $[Z.0 \Rightarrow g(Z.0).R]$
 - = {definition of $Z.0$ }
 - true
- (ii) $j = i+1$ for some i :
 - $[Z.(i+1) \Rightarrow g(Z.(i+1)).R]$
 - = {definition of invariant}
 - $Z.(i+1)$ is an invariant for R
 - = {definition of $Z.(i+1)$ }
 - $\mathcal{A}.(g(Z.i)^\circ.T, R)$ is an invariant for R
 - = {Property 7}
 - true
- (iii) j is a limit ordinal:
 - $[Z.j \Rightarrow g(Z.j).R]$
 - = {definition of $Z.j$ }
 - $[(\exists i: i < j: Z.i) \Rightarrow g(Z.j).R]$
 - = {calculus}
 - $[(\forall i: i < j: Z.i \Rightarrow g(Z.j).R)]$
 - \Leftarrow {calculus}
 - $[(\forall i: i < j: (Z.i \Rightarrow g(Z.i).R) \wedge (g(Z.i).R \Rightarrow g(Z.j).R))]$
 - = {calculus}
 - $[(\forall i: i < j: Z.i \Rightarrow g(Z.i).R)] \wedge [(\forall i: i < j: g(Z.i).R \Rightarrow g(Z.j).R)]$

$$\begin{aligned}
& =\{\text{inductive hypothesis}\} \\
& \quad [(\forall i: i < j: g(Z.i).R \Rightarrow g(Z.j).R)] \\
& \Leftarrow\{\text{left monotonicity of } g(X).Y \text{ and calculus}\} \\
& \quad [(\forall i: i < j: Z.i \Rightarrow Z.j)] \\
& =\{\text{calculus}\} \\
& \quad [(\exists i: i < j: Z.i) \Rightarrow Z.j] \\
& =\{\text{definition of } Z.j\} \\
& \quad \text{true} \tag{End}
\end{aligned}$$

Lemma 6 $[Z.(i+1) \Rightarrow g(Z.i)^\circ.T]$ for all $i \geq 0$.

Proof.

$$\begin{aligned}
& \quad [Z.(i+1) \Rightarrow g(Z.i)^\circ.T] \\
& =\{\text{definition of } Z.(i+1)\} \\
& \quad [\mathcal{A}.(g(Z.i)^\circ.T, R) \Rightarrow g(Z.i)^\circ.T] \\
& =\{\text{Property 8}\} \\
& \quad \text{true} \tag{End}
\end{aligned}$$

Consider again an execution of g with the initial state satisfying $g.R$ and in which g iterates forever. By Lemmas 4 and 5 the state at each invocation of g satisfies $Z.j$ for some $j > 0$ ($j > 0$ because $Z.0 = F$). Hence we may associate with each invocation an ordinal j , namely the least j such that the state at the invocation satisfies $Z.j$. By Lemma 5 this sequence of ordinals is monotonically decreasing. As all strictly decreasing sequences of ordinals are of finite length we infer that eventually all invocations have the same associated ordinal k , for some $k > 0$. It follows from the minimality requirement on the associated ordinals that they are not limit ordinals — because by definition of $Z.m$ for m a limit ordinal, a state satisfying $Z.m$ also satisfies $Z.j$ for some $j < m$. Hence k is a successor ordinal that we may call $i + 1$. We now have an ordinal i such that eventually the state at each invocation satisfies $Z.(i+1)$ but never $Z.i$. We now infer from Lemma 6 that eventually each invocation of g happens in a state satisfying $g(Z.i)^\circ.T$ but never $Z.i$. We may therefore take this $Z.i$ as the Q_0 that was to be exhibited.

5.2 Conjunctionity

To complete the proof of Property 1 in the presence of fairness we shall need:

Lemma 7 $g.X$ is monotonic.

Proof.

$$\begin{aligned}
& \quad [P \Rightarrow Q] \Rightarrow [g.P \Rightarrow g.Q] \\
& =\{\text{detach } [P \Rightarrow Q]\} \\
& \quad [g.P \Rightarrow g.Q] \\
& =\{\text{definition } g.P\} \\
& \quad [(\mu X: \mathcal{A}.(g(X)^\circ.T, P)) \Rightarrow g.Q] \\
& \Leftarrow\{\text{least fixpoint property}\} \\
& \quad [\mathcal{A}.(g(g.Q)^\circ.T, P) \Rightarrow g.Q] \\
& =\{\text{fixpoint property of } g.Q\} \\
& \quad [\mathcal{A}.(g(g.Q)^\circ.T, P) \Rightarrow \mathcal{A}.(g(g.Q)^\circ.T, Q)] \\
& \Leftarrow\{\text{right monotonicity of } \mathcal{A}\} \\
& \quad [P \Rightarrow Q] \\
& =\{\text{detached axiom}\} \\
& \quad \text{true} \tag{End}
\end{aligned}$$

The following lemma separates total correctness of a recursive procedure into partial correctness and a termination argument.

Lemma 8 $[g.R \equiv g.T \wedge \mathcal{A}.(T, R)]$

Proof. Let $h.R = \mathcal{A}.(T, R)$. We will use

$[g.R \Rightarrow h.R]$ (i)
 ={definition of h.R}
 $[g.R \Rightarrow \mathcal{A}.(T,R)]$
 ={fixpoint property of g.R}
 $[\mathcal{A}.(g(g.R)^\circ.T,R) \Rightarrow \mathcal{A}.(T,R)]$
 ={left monotonicity of \mathcal{A} }
 true

For the main proof from left to right

$[g.R \Rightarrow g.T \wedge h.R]$
 ={calculus}
 $[g.R \Rightarrow g.T] \wedge [g.R \Rightarrow h.R]$
 ={Lemma 7, (i)}
 true

For the proof in the other direction we need two preliminary results. Firstly

$[h.R \Rightarrow g(h.R).T]$ (ii)
 \Leftarrow {g(X).Y right monotonic}
 $[h.R \Rightarrow g(h.R).R]$
 ={definitions of invariant}
 h.R an invariant for R
 ={definition of h.R}
 $\mathcal{A}.(T,R)$ an invariant for R
 ={Property 7}
 true

Secondly

$[\mathcal{A}.(h.R,R) \equiv h.R]$ (iii)
 \Leftarrow {Property 9}
 h.R an invariant for R
 ={definition of h.R and Property 7}
 true

To complete the main proof

$[g.T \wedge h.R \Rightarrow g.R]$
 ={calculus}
 $[g.T \Rightarrow \neg h.R \vee g.R]$
 ={definition of g.T}
 $[(\mu X:\mathcal{A}.(g(X)^\circ.T,T)) \Rightarrow \neg h.R \vee g.R]$
 \Leftarrow {least fixpoint property}
 $[\mathcal{A}.(g(\neg h.R \vee g.R)^\circ.T,T) \Rightarrow \neg h.R \vee g.R]$
 ={calculus}
 $[\mathcal{A}.(g(\neg h.R \vee g.R)^\circ.T,T) \wedge h.R \Rightarrow g.R]$
 ={(iii)}
 $[\mathcal{A}.(g(\neg h.R \vee g.R)^\circ.T,T) \wedge \mathcal{A}.(h.R,R) \Rightarrow g.R]$
 ={conjunctivity of \mathcal{A} }
 $[\mathcal{A}.(g(\neg h.R \vee g.R)^\circ.T \wedge h.R, R) \Rightarrow g.R]$
 \Leftarrow {(ii), left monotonicity of \mathcal{A} , calculus}
 $[\mathcal{A}.(g(\neg h.R \vee g.R)^\circ.T \wedge g(h.R).T, R) \Rightarrow g.R]$
 \Leftarrow {Property 19, left monotonicity of \mathcal{A} , calculus}
 $[\mathcal{A}.(g(h.R \wedge g.R)^\circ.T, R) \Rightarrow g.R]$
 ={(i), calculus}
 $[\mathcal{A}.(g(g.R)^\circ.T, R) \Rightarrow g.R]$
 ={fixpoint property of g.R}
 true

(End)

Proof of Property 1 completed:

$g.X \wedge g.Y$

$$\begin{aligned}
&= \{\text{Lemma 8}\} \\
&\quad g.T \wedge \mathcal{A}.(T,X) \wedge g.T \wedge \mathcal{A}.(T,Y) \\
&= \{\text{right conjunctivity of } \mathcal{A}\} \\
&\quad g.T \wedge \mathcal{A}.(T,X \wedge Y) \\
&= \{\text{Lemma 8}\} \\
&\quad g.(X \wedge Y) \qquad \qquad \qquad (\text{End})
\end{aligned}$$

Proof of Property 19 completed:

With the introduction of recursion there is an extra case to consider in the proof of Property 19, namely that for h a procedure $[h^\circ.R \wedge h.S \Rightarrow h^\circ.(R \wedge S)]$. The proof proceeds just like the proof of Lemma 8 from right to left, but appealing to the invariance property of $h.S$ rather than its fixpoint property. We leave it to the reader. (End)

Programs enjoy the so-called law of the excluded miracle [2], and continue to do so in the presence of fairness:

Property 20 [$s.F \equiv F$]

Proof. By structural induction, the only novel case being that of $g.G$.

$$\begin{aligned}
&[g.F \equiv F] \\
&= \{\text{calculus}\} \\
&[g.F \Rightarrow F] \\
&= \{\text{definition}\} \\
&[(\mu X: \mathcal{A}.(g(X)^\circ.T, F)) \Rightarrow F] \\
&\Leftarrow \{\text{least fixpoint property}\} \\
&[\mathcal{A}.(g(F)^\circ.T, F) \Rightarrow F] \\
&= \{\text{definition of } \mathcal{A}\} \\
&[(\eta X: g(F)^\circ.T \wedge g(X).F) \Rightarrow F] \\
&= \{\text{fixpoint property}\} \\
&[g(F)^\circ.T \wedge g(\dots).F \Rightarrow F] \\
&\Leftarrow \{\text{Property 19}\} \\
&[g(F)^\circ.F \Rightarrow F]
\end{aligned}$$

So we are done if programs without fairness but with angelic nondeterminacy enjoy the law of the excluded miracle. That is known to be true, and can be shown by structural induction. (End)

There is a property of programs that is lost with the introduction of fairness, that of "bounded nondeterminacy"; it will be considered in section 7.

6 Variant Functions

It is usual to reason about loops and recursion using the notions of an invariant assertion and a variant function, and this idea carries forward into programs with fairness. As an introduction, and for purposes of comparison with what is to follow, we state a version of the well-known invariance theorem in our notation. The "arithmetical" operators $<, \leq, =$ etc. have precedence above the logical operators.

Theorem 1 *Let g be a tail-recursive procedure without fair choice, P and R be assertions, (C, \leq) be a well-founded set, and t be an expression on the state space. If*

$$\begin{aligned}
&[P \Rightarrow t \in C] \\
&[P \Rightarrow g(P).R] \\
&[P \wedge t = x \Rightarrow g(t < x).T] \text{ for all } x \in C
\end{aligned}$$

then $[P \Rightarrow g.R]$

(The reader should recognise P as the invariant assertion and t as the variant function.) (End)

The theorem for weak fairness accounts for termination with a variant function that can never be increased and which may possibly be decreased:

Theorem 2 *Let g be a tail-recursive procedure, P and R be assertions, (C, \leq) be a well-founded set, and t be an expression on the state space such that*

$$[P \Rightarrow t \in C] \quad (i)$$

$$[P \Rightarrow g(P).R] \quad (ii)$$

$$[P \wedge t = x \Rightarrow g(t \leq x).T] \text{ for all } x \in C \quad (iii)$$

$$[P \wedge t = x \Rightarrow g(t < x)^\circ.T] \text{ for all } x \in C \quad (iv)$$

then $[P \Rightarrow g.R]$

Proof. First we manipulate (ii), (iii), and (iv) into the more usable

$$[P \wedge t \leq x \Rightarrow g(P \wedge t \leq x).R] \text{ for all } x \in C \quad (v)$$

$$[P \wedge t \leq x \Rightarrow g(P \wedge t < x)^\circ.T] \text{ for all } x \in C \quad (vi)$$

Firstly

$$\begin{aligned} & (v) \\ =\{ & g(X).Y \text{ left conjunctive} \} \\ & [P \wedge t \leq x \Rightarrow g(P).R \wedge g(t \leq x).T] \\ \leftarrow \{ & \text{calculus} \} \\ & [P \Rightarrow g(P).R] \wedge [P \wedge t \leq x \Rightarrow g(t \leq x).T] \\ =\{ & (ii) \} \\ & [P \wedge t \leq x \Rightarrow g(t \leq x).T] \\ =\{ & \text{calculus, in particular the one-point rule} \} \\ & [(\forall c:: c \leq x \wedge P \wedge t = c \Rightarrow g(t \leq x).T)] \\ \leftarrow \{ & (iii) - c \in C \text{ from } P \wedge t = c \text{ and (i)} \} \\ & [(\forall c:: c \leq x \wedge g(t \leq c).T \Rightarrow g(t \leq x).T)] \\ \leftarrow \{ & g(X).Y \text{ left monotonic} \} \\ & [(\forall c:: c \leq x \wedge g(t \leq c).T \Rightarrow g(t \leq x).T)] \\ =\{ & \text{calculus} \} \\ & \text{true} \end{aligned}$$

The proof of (vi) proceeds similarly and is left to the reader. We shall be using well-founded induction in a form which is explained in a note below. The proof of the theorem is:

$$\begin{aligned} & [P \Rightarrow g.R] \\ =\{ & (i) \} \\ & [P \wedge t \in C \Rightarrow g.R] \\ =\{ & \text{inductive hypothesis: introduce } x \text{ such that } x \in C \wedge [P \wedge t \in C \wedge t < x \Rightarrow g.R] \} \\ & [P \wedge t = x \Rightarrow g.R] \\ \leftarrow \{ & \text{calculus} \} \\ & [P \wedge t \leq x \Rightarrow g.R] \\ =\{ & \text{fixpoint property} \} \\ & [P \wedge t \leq x \Rightarrow \mathcal{A}.(g(g.R)^\circ.T, R)] \\ =\{ & \text{Property 9, (v)} \} \\ & [\mathcal{A}.(P \wedge t \leq x, R) \Rightarrow \mathcal{A}.(g(g.R)^\circ.T, R)] \\ \leftarrow \{ & \text{left monotonicity of } \mathcal{A} \} \\ & [P \wedge t \leq x \Rightarrow g(g.R)^\circ.T] \\ \leftarrow \{ & (vi) \} \\ & [g(P \wedge t < x)^\circ.T \Rightarrow g(g.R)^\circ.T] \\ \leftarrow \{ & g(X)^\circ.T \text{ left monotonic} \} \\ & [P \wedge t < x \Rightarrow g.R] \\ =\{ & \text{inductive hypothesis, (i)} \} \\ & \text{true} \end{aligned}$$

(End)

Note on well-founded induction. Well-founded induction in its familiar form is a theorem that facilitates proofs of formulae of the shape $(\forall x: x \in C: Q.x)$ where (C, \leq) is a well-founded set and $Q.x$ is a predicate. The theorem states:

$$(\forall x: x \in C: Q.x) \equiv (\forall x: x \in C: (\forall y: y \in C \wedge y < x: Q.y) \Rightarrow Q.x).$$

There is another form of well-founded induction for proving formulae of the shape $(\forall x: f.x \in C: Q.x)$ where f is any function (whose range may or may not be a subset of C):

$$(\forall x: f.x \in C: Q.x) \equiv (\forall y: y \in C: (\forall x: f.x \in C \wedge f.x < y: Q.x) \Rightarrow (\forall x: f.x = y: Q.x)).$$

Letting $Q.x$ have the form $P.x \Rightarrow Q.x$ we easily rewrite the preceding as:

$$\begin{aligned} & (\forall x: P.x \wedge f.x \in C \Rightarrow Q.x) \equiv \\ & (\forall y: y \in C \wedge (\forall x: P.x \wedge f.x \in C \wedge f.x < y \Rightarrow Q.x) \Rightarrow (\forall x: P.x \wedge f.x = y \Rightarrow Q.x)), \end{aligned}$$

and this is the kind of well-founded induction used in the proof of the above theorem. This is a frightening looking formula, but it is easy to use in practice: When we find ourselves confronted with proving $(\forall x: P.x \wedge f.x \in C \Rightarrow Q.x)$ we just postulate the truth of $y \in C \wedge (\forall x: P.x \wedge f.x \in C \wedge f.x < y \Rightarrow Q.x)$ — the induction hypothesis — and proceed with proving $(\forall x: P.x \wedge f.x = y \Rightarrow Q.x)$ feeling free to appeal to the inductive hypothesis as we need. See [6]. (End)

Observe that Theorem 2 reduces to Theorem 1 in the absence of fairness. It is sometimes easier to use (v) and (vi) instead of (ii), (iii), and (iv) when applying Theorem 2.

Example 4 Consider

g1: skip []_f i:= i+1; g1

To apply Theorem 2, for P and R take T , for (C, \leq) take the natural numbers with the usual ordering, and for t take the constant function which maps all states to 0, say. Then conditions (i) to (iv) reduce to showing $[T \Rightarrow g1(T).T]$ — that T is an invariant — and $[T \Rightarrow g1(F)^\circ.T]$ — that $g1$ may terminate on each iteration; these are evidently true and their formal verification is routine. Hence $[g1.T \equiv T]$. (End)

Example 5 Consider

g2: g2 []_f if i > 0 then i:= i-1; g2 else skip fi

We can apply Theorem 2 to infer $[g2.T \equiv T]$. For invariant take T , and for variant function take $\text{abs}.i$, where abs is the function that returns the absolute value of its integer argument; $\text{abs}.i$ cannot be increased and may always be decreased until $i = 0$. (End)

Example 6 Consider

**g3: if b then (i:= i+1 []_f b:= false); g3
else g2 {of Example 5} fi**

We want to infer $[g3.T \equiv T]$. Choose invariant T , and variant function $t = \text{abs}.i + (\omega \times h.b)$ where ω denotes the first infinite ordinal and h maps the booleans to $\{0, 1\}$ by $h.\text{false} = 0$ and $h.\text{true} = 1$. While b is true t has the value ω regardless of the value of i ; moreover, the assignment $b := \text{false}$, which is always possible while b holds, reduces t to a finite value which thereafter can only be decreased. It can be proved that the natural numbers do not suffice for proving termination in this case — we have taken for C an infinite ordinal, say ω^2 . (End)

Example 7 Consider

```

g4:  if i = 10 then skip
      else i:= 0; g1 {of Example 4}; g4 fi

```

(Note that the presence of $g1$ does not contravene the requirement of $g4$'s recursion being tail recursion; the requirement is that all recursive calls of $g4$ are dynamically last calls, and that requirement is met.) Using the invariant T and the variant function $h.(i \neq 10)$ as defined in the preceding example we may infer $[g4.T \equiv T]$. We will return to this example later. (End)

7 Characteristics of fairness

The choice operator is evidently commutative and associative, so it is reasonable to apply it to a bag of statements. We will write

$$(\coprod j: j \in S: s.j) \tag{1}$$

to denote choice applied to all the $s.j$'s such that $j \in S$. For example

$$(\coprod j: j \in \text{nat}: i:=j) \tag{2}$$

where nat denotes the set of natural numbers is equivalent to $i:=0 \sqcup i:=1 \sqcup i:=2 \sqcup \dots$. We define $(\coprod_f j: j \in S: s.j)$ analogously. If S is bounded then (1) can obviously be expressed as a program (without fairness) if each $s.j$ can, but if S is unbounded then (1) has no equivalent program using traditional semantics; we say that programs have the property of "bounded nondeterminacy". This property no longer holds with the introduction of fairness. Indeed (2) has the same predicate transformer semantics as $i:=0; g1$ where $g1$ is given in Example 4; this is easy to verify and is left to the reader. However, if $i:=0; g1$ in Example 7 is replaced with (2) then $g4.T$ evaluates not to T but to $i=10$, the reason being that $i:=0; g1$ implements not (2) but $(\coprod_f j: j \in \text{nat}: i:=j)$. Fairness as we have defined it introduces not unbounded nondeterminacy but unbounded *fair* nondeterminacy. This will be seen to be significant when we consider implementation issues.

We have two kinds of nondeterminacy represented by \sqcup and \sqcup_f . The essential difference between them is, to borrow terminology from Park[7], that \sqcup captures "loose" nondeterminacy whereas \sqcup_f captures "tight" nondeterminacy. Loose nondeterminacy requires simply that every result of a nondeterministic computation be one of those prescribed by the semantics; tight nondeterminacy requires in addition that all the admissible results are in fact possible outcomes of the computation. We should expect a customer to raise no objections if we replace $i:=1 \sqcup i:=2$ in his or her program with $i:=1$ – but replacing $i:=1 \sqcup_f i:=2$ with $i:=1$ may well be fatal:

Example 8

```

g5:  if j=0 then skip
      else
          (i:=1  $\sqcup_f$  i:=2);
          if i=j then j:=j-1 else skip fi;
      g5
      fi

```

We can easily show (using Theorem 2 with variant j) that $j:=2; g5$ always terminates. However, it will iterate forever if $i:=1 \sqcup_f i:=2$ is replaced with $i:=1$ or $i:=2$. (End)

It seems clear, however, that termination of a recursive procedure cannot depend on a fair component delivering an unbounded number of outcomes; however, the particular bounded sequence of outcomes that leads to termination may depend on the computation.

A consequence of the tight nondeterminacy of fair choice is the loss of a property of programs that we shall call "monotonic replacement". Let us introduce relation \sqsubseteq on programs such that $s \sqsubseteq t$ equals "program s is refined by (or is implemented by, or may be replaced by) program t ". Formally $s \sqsubseteq t \equiv (\forall X :: [s.X \Rightarrow t.X])$. Monotonic replacement is the law that for all programs u , v , and s , with u possibly occurring in program s , $u \sqsubseteq v \Rightarrow s \sqsubseteq s(u/v)$ – in words, that we can refine a program by refining a component of it in isolation. It is a crucial law in the theory of program construction. Now $(i:=1 \parallel_f i:=2) \sqsubseteq i:=1$ but it is not the case that $g5$ of Example 8 is refined by replacing $i:=1 \parallel_f i:=2$ by $i:=1$. We can continue to apply monotonic replacement, of course, as long as we do not reduce - or at least are very careful about reducing - the degree of fair nondeterminacy.

A final characteristic of fairness is the well-known fact that fair computations are of unbounded length: we cannot in general place an upper bound on the number of computation steps taken by a fair computation before it terminates.

8 Bounded Weak Fairness

With fairness as we have defined it, a recursive procedure not only uses fair choice to terminate, but makes that fairness externally visible. In Example 4, for example, $g1$ uses fairness to terminate – but procedures invoking $g1$, such as $g4$ in Example 8, may rely on the fairness not just for $g1$'s termination, but for its termination in a favourable state. We consider now a kind of fairness which by contrast is purely local, without global consequences. We retain the definition of section 5 but with a revised meaning for $g(X)^\circ.T$. Previously we defined the $^\circ$ operation as one of replacing \parallel_f 's with \parallel_a 's; for the new fairness we restrict the replacement to \parallel_f 's not occurring in nested procedures. More precisely, in a definition of $g(X)^\circ$ by structural induction we define f° for f a procedure invocation by $f^\circ = f$. The semantics of the procedures of Examples 4 and 5 is not changed by the new definition - because they contain no nested procedures, and the reader will also convince him-/herself that $g3$ of Example 4 continues to behave as before. But $g4$ behaves differently:

Example 9 We show that now $[g4.T \equiv i=10]$ where $g4$ is the procedure of Example 7. That $[i=10 \Rightarrow g4.T]$ is obvious. For the other direction we first show $[g1.(i=10) \equiv F]$.

$$\begin{aligned} & g1.(i=10) \\ =\{ \text{Lemma 8} \} & g1.T \wedge \mathcal{A}_{g1}.(T, i=10) \\ =\{ \text{Example 4, remarks in preceding paragraph} \} & \mathcal{A}_{g1}.(T, i=10) \\ =\{ \text{definition of } \mathcal{A} \} & (\eta X: g1(X).(i=10)) \end{aligned}$$

Let $f.X = g1(X).(i=10)$. We leave it to the reader to show $[f.T \equiv i=10]$, and then $[f.(f.T) \equiv F]$. Hence

$$\begin{aligned} & [g1.(i=10) \equiv F] \\ =\{ \text{preceding} \} & [(\eta X: f.X) \equiv F] \\ =\{ \text{fixpoint twice} \} & [f.(f.(\eta X: f.X)) \equiv F] \\ =\{ f \text{ monotonic, } [f.(f.T) \equiv F] \} & \text{true} \end{aligned}$$

Knowing $[g1.(i=10) \equiv F]$ we may easily calculate $[g4^\circ(i=10).T \equiv i=10]$. Hence

$$\begin{aligned} & [g4.T \Rightarrow i=10] \\ \Leftarrow \{ \text{least fixpoint property of } g4.T \} & [\mathcal{A}_{g4}.(g4^\circ(i=10).T, T) \Rightarrow i=10] \\ =\{ \text{see above} \} & [\mathcal{A}_{g4}.(i=10, T) \Rightarrow i=10] \\ =\{ \text{Property 8} \} & \end{aligned}$$

true

(End)

A consequence of the new kind of fairness is that unbounded nondeterminacy is no longer fair. Now, $i:=0; g1$ is equivalent to $(\prod j: j \in \text{nat}: i:=j)$ rather than $(\prod j: j \in \text{nat}: i:=j)$. Hence we call this fairness "bounded weak fairness".

Bounded weak fairness preserves monotonic replacement at the procedure level: for procedures g and f , and program s we have $g \sqsubseteq f \Rightarrow s \sqsubseteq s(g/f)$ – we leave the verification of this to the reader.

Bounded weak fairness was defined by hiding the fairness in *all* constituent procedures. We could be selective, however, by giving the programmer the option of hiding or not the fairness at each procedure invocation – the mathematics does not object.

9 A Strong Fairness

It may be overly demanding to require that an execution of g should be able to make progress on every iteration. It suffices to require that the opportunity to progress comes repeatedly, and this is the motivation behind what we now introduce as "strong fairness". Weak fairness is a special case of strong fairness. We replace the definition of section 5 with

$$g.R = (\mu X: \mathcal{I}.(g(X)^\circ.T, R))$$

Its justification proceeds much as for weak fairness: the difference is that in the event of infinite iteration we can exhibit a predicate $Q0$ such that $g(Q0)^\circ.T$ holds on *infinitely many* invocations of g , but never $Q0$. Again g is conjunctive and satisfies $[g.F \equiv F]$. The proofs proceed much as for those of weak fairness, using the results of subsection 3.4, and are omitted. The use of variant functions for proving termination is more complicated: we use two functions t and u such that each iteration cannot increase t and either decrements u or admits the possibility of decrementing t .

Theorem 3 *Let g be a tail-recursive procedure, P and R be assertions, (C, \leq) and (D, \leq) be well-founded sets, and t and u be expressions on the state space such that*

$$[P \Rightarrow t \in C \wedge u \in D]$$

$$[P \Rightarrow g(P).R]$$

$$[P \wedge t=x \Rightarrow g(t \leq x).T] \text{ for all } x \in C$$

$$[P \wedge t=x \wedge u=y \Rightarrow g(t < x)^\circ.T \vee g(u < y).T] \text{ for all } x \in C, y \in D$$

then $[P \Rightarrow g.R]$

Proof. The proof is much the same as that of Theorem 2 and is omitted.

(End)

Strong fairness admits the same variations as weak fairness, as described in the preceding section.

Example 10 Consider

```

g6:  if i > 0 then
      if ¬b then i:=i-1 else b:=¬b fi; g6
    else skip fi
    [] b:=¬b; g6

```

We apply Theorem 3 to infer $[g.T \equiv T]$. For invariant take T . For t take $\text{abs}.i$ and for u take $\text{h}.b$ as defined in Example 6. It is evident that $\text{abs}.i$ cannot be increased. Furthermore, when $\text{h}.b = 1$ either the procedure terminates or $\text{h}.b$ is decreased, and when $\text{h}.b = 0$ then $\text{abs}.i$ may be decreased or the procedure may terminate.

(End)

10 Implementation Issues

Can fairness be implemented? More precisely, can we make an implementation that terminates when presented with a procedure g , containing fair choice, in initial state $g.T$. The component that interests us is the arbiter. We confine our attention for the moment to weak fairness. By the argument of subsection 5.1 an implementation fails to terminate by failing to establish a predicate for which it has the opportunity on every iteration. The arbiter is never faced with a bad choice, only neutral and good ones. At first sight it might seem an adequate implementation simply to impose on the arbiter the obligation to "try all", i.e. to follow all the execution paths of g systematically and periodically. Assuming the implementation can periodically enumerate the possibilities – and the presence of unbounded fairness doesn't exclude this – then we might expect steady progress towards termination. It would be a hit-and-miss affair of course, but the misses do no harm and the hits should come regularly enough. This try-all strategy seems adequate for the procedures of Examples 4, 5 and 6, for example, but sadly it does not suffice in general:

Example 11

```
g7:  if b then
      (b:= ¬c; c:= true []f b:=c; c:=false); g7
      else skip fi
```

An execution of g_6 with initially $b = \text{true}$ terminates by the semantics of weak (bounded) fairness. Yet an implementation that cycles through the possibilities by selecting alternately the left and right components of the fair choice either terminates after one iteration or fails to terminate. (End)

The try-all implementation having failed, we resort to probabilistic methods. Let us require of the arbiter that it give to each nondeterministic choice in statements that are to be treated fairly a positive probability of being selected. Assume for the moment that the fairness is a bounded type; we can then further insist that the probabilities be non-vanishing. Hence in an execution of procedure g , a favourable action – one that brings progress – will eventually be selected with probability one, and so g terminates almost surely (i.e. with probability one). If the fairness is unbounded then the probability of progressing towards termination with this strategy is in general vanishing, and so the probability of terminating is less than 1. This is a pity but there seems no escaping it.

Strong fairness may be implemented by the same probabilistic strategy as for weak fairness provided the fairness is bounded: if all choices can be given a non-vanishing probability of being selected then, the opportunity to progress coming infinitely often, progress will eventually be made with probability one.

But there is a sense in which it is legitimate to say that fairness is not implementable. For to implement fairness we must construct an arbiter, and once we fix the arbiter the fairness vanishes: knowing the arbiter we could rewrite the program without using any fair choice and without any reduction in nondeterminacy over what the agreed arbiter would provide. Furthermore fairness as such is not in a certain sense useful: a guarantee of termination is not of practical significance if we have no upper bound on the length of the computation. If fairness is of any benefit it is because it is an abstraction: it abstracts from the details of arbiters allowing us to reason more easily about termination, it being a further, but separate, concern to calculate upper bounds on the length of the computation knowing the details of the arbiter actually used. What we have shown, in the present case, is that the class of arbiters accommodated by our semantics is a class of probabilistic ones.

The fact that our fair semantics admits only probabilistic arbiters comes as a little surprise. One of the motivations behind the present approach was to construct a semantics of fairness in which the syntactic shape of the program was not significant. In contrast, many treatments of fairness begin by fixing the shape of the program, typically as **do** $b_0 \rightarrow s_0$ [] $b_1 \rightarrow s_1$ [] $b_2 \rightarrow s_2$ [] ... **od**. In retrospect, the

fact that only probabilistic arbiters are accommodated stems in large measure from our decision to ignore the shape of the program. Probabilistic arbiters are used in practice to implement fairness, for example in CSMA/CD network protocols, but an implementation that is fair by virtue of cycling through the possibilities is more commonly assumed. There seems no reason why we shouldn't be able to use the machinery of temporal predicate transformers to describe such a fairness, perhaps at the price of exploiting the syntactic shape of the program, but we will not try to substantiate that claim here.

11 Summary and Conclusion

There is an extensive literature on fair termination, comprehensively surveyed in [4]. Two notions of fairness have predominated: weak fairness which requires that "every action that is continuously enabled is eventually taken", and strong fairness which requires that "every action that is infinitely often enabled is infinitely often taken". Much of the work on fair termination has been concerned with giving formal expression to these operational ideas, many variations coming about because of the variety of interpretations that can be given to "action" and "enabled". The distinctive features of our approach are as follows. Firstly, whereas fair termination of programs has usually been formulated as Hoare-style proof rules the present treatment uses the machinery of predicate transformers and fixpoint theory. We proceeded by formulating predicate transformers to capture the temporal notions of *always*, *eventually*, and *infinitely often*, and used these as the building blocks of our theory of fairness. Secondly, we have not so much been concerned with specific definitions of fairness, although we have given some, but with creating a framework that accommodates many notions of fairness uniformly. Thirdly, our fairness is not confined to guarded commands, but is denoted by a fair choice operator that may be employed arbitrarily in the program. Indeed we do not exclude arbitrary choice and various kinds of fair choice coexisting in the same program. Fourthly, our notions of fairness do not exploit the syntactic shape of programs – for example by attaching counters to guards that are decremented each time the guard is ignored and which eventually ensure priority for the guarded command. Fifthly, we have based our treatment on recursion rather than on loops, the latter being preferred heretofore. Sixthly, the kinds of fairness we have introduced admit implementations only with probabilistic arbiters, and provided the fairness is bounded. The machinery we have introduced appears to be equally capable of describing the kind of fairness that cycles through the choices, provided we are willing to exploit the syntactic shape of the program.

The main advantages of the present approach seems to be that it provides a framework for accommodating a range of fairness notions with ease, and that it imposes but modest proof obligations on the programmer.

Acknowledgements. I began this work at University College, Dublin, continued it during a stay at the Technische Universiteit, Eindhoven, and completed it at the University of Glasgow; I thank these institutions for the facilities they have given me. I particularly thank W. H. J. Feijen for arranging my visit to Eindhoven; the paper is very much a record of my education, for which he was largely responsible, during that visit. The calculational proof style I have employed is due to him. The members of the Eindhoven Tuesday Afternoon Club provided a scientific forum to which it was a pleasure to belong. I thank Jaap van der Woude for much enlightenment on the theory of lattices and fixpoints. C. J. van Rijsbergen arranged my visit to Glasgow and provided me with a good working environment, for which I thank him. Technically, I have drawn much inspiration from [3]: primarily, I have taken the extensive use of fixpoint properties, the importance of conjunctivity properties, the guts of Lemma 2, and the idea of Lemma 8. The benefits of using fixpoints to describe the weakest precondition semantics of iteration was apparently first pointed out by Park[7]. Sections 7 and 10 have drawn on an insightful discussion of fairness in [7], to which one of the referees drew my attention. I am grateful to the referees for their very helpful criticisms which have led to considerable clarification.

References

1. Cousot, P. and Cousot, R.: Constructive versions of Tarski's fixed point theorems. *Pac. J. Math.* **82**, 43-57 (1979)
2. Dijkstra, E. W.: *A discipline of programming*. Englewood Cliffs, N. J.: Prentice-Hall 1976
3. Dijkstra E. W.: Extreme solutions of equations. Univ. of Texas at Austin, EWD969, Sept. 1986
4. Francez, N.: *Fairness*. Berlin Heidelberg New York: Springer-Verlag 1986
5. Morris, J. M.: A theoretical basis for stepwise refinement and the programming calculus. *Sci. Comput. Program.* **9**, 287-306 (1987).
6. Morris, J. M.: Well-founded induction and the invariance theorem for loops. *Inf. Process. Lett.* to appear.
7. Park, D.: On the semantics of fair parallelism. In: Bjørner, D. (ed) *Abstract Software Specifications. Proceedings, Copenhagen 1979*. (Lect. Notes in Comput. Sci, vol. 86, pp. 504-526) Berlin Heidelberg New York: Springer-Verlag 1980
8. Tarski, A.: A lattice theoretical fixpoint theorem and its applications. *Pac. J. Math.* **5**, 285-309 (1955)