Theoretical Computer Science 33 (1984) 65–84
North-Holland

FAIR TERMINATION REVISITED—WITH DELAY*

K.R. APT
LITP, Université Paris 7, 75251 Paris, France

A. PNUELI
The Weizmann Institute of Science, Rehovot, Israel

J. STAVI
Bar-Ilan University, Ramat Gan, Israel

Abstract. A proof method for establishing the fair termination and total correctness of both nondeterministic and concurrent programs is presented. The method calls for the extension of states by auxiliary delay variables which count down to the instant in which a certain action will be scheduled. It then uses well-founded ranking to prove fair termination allowing nested fair selection and loops.

1. Introduction

The problem of termination of nondeterministic and concurrent programs under the assumption of fairness has recently been receiving considerable attention (see, e.g., [1, 6, 9]).

The basic method for proving invariant properties, such as partial correctness, was developed by Floyd [5] and Hoare [8] for sequential programs. It is based on the idea of finding an inductive property which is preserved by every basic action of the program. When we consider nondeterministic and concurrent programs, the method of invariance is still applicable with very minor modifications.

In comparison, the suggested method for proving termination properties (total correctness for example [10]) is not directly extendable to concurrent and nondeterministic programs when we stipulate fair executions. The method, as developed in [5, 11] is based on establishing a mapping from the program states to some well-founded domain (a rank) such that any program action causes a decrease in the rank. That this method does not apply to fair termination is obvious from the

* The work reported here was partly done while the first author was visiting the Weizmann Institute, Rehovot, Israel.
It is supported in part by the Israeli Academy of Sciences, Basic Research Foundation.

following trivial example:

```
while $b$ do if [$b \rightarrow$ skip $\square$ $b \rightarrow b :=$ false] fi od.
```

A fair execution of this program must eventually choose the second branch of the conditional, causing b to be set to **false** and terminating the program. However, any choice of the first branch preserves the program state. Correspondingly, no mapping which *always* causes a decrease in the rank can exist.

The study of fair executions is mainly motivated by concurrent programs. For concurrent computations fairness or its weaker version—justice [9]—is a most general modelling of the fact that the ratio of speeds between cooperating processors may be arbitrarily large and varying but is always finite. The study of fairness in the context of nondeterministic but sequential programs is motivated in part by the use of nondeterminism to model concurrency and also as a more restricted interpretation of nondeterminism.

Answering to the challenge of extending the method of well-founded orderings to fair termination, several suggestions were made.

One approach, represented by [9] and [6] is to relax the requirement that *every* action causes a decrease in the rank of the state. By this methodology, for each state there always exist some *helpful* actions which decrease the rank of the state, and some other actions, termed *indifferent* (steady in [6]), which at least do not increase this rank. By fairness (and some additional requirements of the method), a helpful action must eventually be chosen which causes the rank to decrease and thus excludes infinite computations. This method was applied in [9] to concurrent programs represented in an abstract framework, and in [6] to nondeterministic programs in a more syntax directed style. An interesting point is that the method of [6] can only be applied to programs which terminate due to fairness on the top level, i.e., fair choice between the branches of an encompassing loop and not between branches of an enclosed conditional statement.

Thus the following example,

```
while $b$ do
  if [$b \rightarrow$ skip
    $\square b \rightarrow$ if [$b \rightarrow$ skip
              $\square b \rightarrow b :=$ false] fi
    ] fi od,
```

cannot be proven fairly terminating by the method of [6].

Another approach to fair termination developed in [1] suggests modifying the program by the construction of an explicit fair scheduler for the program. This reduces the problem of fair termination to that of the termination of a deterministic program in which random assignments $x := ?$ of unbounded natural numbers are allowed. Such assignments are used by the scheduler to implement fair scheduling. By [2] the termination of such programs can always be proved by well-founded ranking, provided we allow ordinals higher than ω—the first countable ordinal.

Once the proof rules are obtained for the program augmented by the scheduler statements, these statements can be eliminated. Thus, we do not have to actually construct the scheduler in order to apply the derived proof rules. They are directly applicable to the program as originally presented. In [1] this method was developed again only for top level fairness in nondeterministic programs.

In this paper we present another approach to the termination of fair programs, covering both concurrent and nondeterministic programs. We believe it to be much simpler and more natural than any of the approaches discussed above, and as we will illustrate below, directly applicable. While the method can also be justified by program transformations, as in [1], the presented justification does not call for program modification but instead extends the states by adding auxiliary variables. In a certain sense this extension parallels the introduction of auxiliary variables in [12] providing a natural method for invariance properties of concurrent programs. As will be shown below our method provides proofs for termination under the assumption of *overall* fairness and not only top level fairness. Thus, in comparison with previous proof methods the approach suggested here is more general, is simpler to apply and justify and forms a natural generalization of the method of well-founded ranking successfully used for sequential programs.

Similarly to [1] we will show that the problem of fair total correctness of a nondeterministic program is reducible to that of the ordinary total correctness of a program which allows random assignments $x := ?$. Such programs were studied in [2]. In our paper we will show the following additional result concerning such reductions in the other direction.

Given a program Π which allows random assignments, it is possible to construct a nondeterministic program Π_1 with no random assignments such that the fair total correctness of Π_1 is equivalent to the ordinary total correctness of Π. Furthermore, it is sufficient to require top level fairness in the computations of Π_1. This result allows us to resolve the issue raised in [1] by showing that all recursive ordinals (order types of recursive well ordering of sets of natural numbers) are required to establish fair termination of programs with top level fairness only. This of course is a significant increase in complexity over the sequential deterministic case where ω is the highest ordinal ever needed.

2. Concurrent programs

The method is illustrated first for concurrent programs represented in an unstructured framework. The framework is taken from [9] and we repeat its basic definitions here:

A concurrent system is a triple:

$$P = \langle S, F, I \rangle,$$

where S is a set of execution states, $I \subseteq S$ is the set of initial states, and $F = \langle f_1, \ldots, f_m \rangle$ is a set of transition functions associated with m processes.

Each $f_i: S \to 2^S$ maps a state s into a set $f_i(s) \subseteq S$ which is the set of possible outcomes when the process P_i executes an atomic instruction on the state s.

If $f_i(s) \neq \emptyset$ we say that f_i is *enabled* on s, otherwise we say that it is *disabled on* s. A state s which is disabled for all $i = 1, \ldots, m$ is called *terminal*. Let T denote the set of terminal states.

An *execution sequence* of P is a maximal sequence

$$s_0 \to^{f_{i_1}} s_1 \to^{f_{i_2}} s_2 \to \cdots,$$

such that $s_0 \in I$ and for each j, $s_{j+1} \in f_{i_{j+1}}(s_j)$. A state is *accessible* if it occurs in an execution sequence. The set of accessible states is denoted by $\mathrm{Acc}(I)$.

An execution sequence is *fair* if it is either finite or if every transition f_k which is enabled infinitely many times in the sequence is also scheduled infinitely many times, i.e., $i_j = k$ for infinitely many j's.

We say that a program P is *fairly convergent* if every fair execution sequence of P is finite.

We propose the following proof method for proving the fair convergence of concurrent systems. By an *extended state* we mean an element of $S \times N^m$.

The delay variables method

(1) Choose a state predicate $Q \subseteq S$ such that

(A) $s \in I \Rightarrow (s \in T) \vee Q(s)$.

(B) $Q(s) \wedge s' \in f_i(s) \Rightarrow (s' \in T) \vee Q(s')$ for $i = 1, \ldots, m$

(T being the set of terminal states).

This ensures that the predicate Q holds for all accessible nonterminal states.

(2) Choose a well-founded set $(W, >)$, i.e., a set W with an *ordering* relation $>$ such that every W-sequence $w_0 > w_1 > \cdots$, $w_i \in W$, is finite.

(3) Find a ranking function

$$\rho : S \times N^m \to W,$$

mapping extended states into the well-founded domain W. An extended state consists of a state $s \in S$ augmented by m *scheduling* (or *delay*) variables $z_1, \ldots, z_m$ (also referred to as *counters*). The role of the delay variable z_i is to count how many steps will pass in which f_i is enabled but not yet scheduled. By fairness, there can be only a finite number of them.

The ranking function must satisfy

$$Q(s) \wedge s' \in f_i(s) \wedge \bigwedge_{j \neq i} [(f_j(s) \neq \emptyset \Rightarrow z_j = z_j' + 1) \wedge (f_j(s) = \emptyset \Rightarrow z_j = z_j')] \Rightarrow$$

$$\Rightarrow \rho(s, z_1, \ldots, z_{i-1}, 0, z_{i+1}, \ldots, z_m) > \rho(s', z_1', \ldots, z_m').$$

All the free variables in the above, i.e., s, s', $\bar{z}$, $\bar{z}'$ are considered to be universally quantified. To prove correctness (or soundness) of the method, consider an infinite fair execution sequence

$$\sigma : s_0 \to^{f_{i_1}} s_1 \to^{f_{i_2}} s_2 \to \cdots.$$

For each $j = 0, 1, \ldots$ we define the vector of delay values $\bar{u}^j = (u_1^j, \ldots, u_m^j) \geq 0$ as follows: u_k^j = number of distinct $j' \geq j$ such that f_k is enabled on s_j, but not scheduled for any $j \leq l \leq j'$, i.e., number of contiguous steps from j on where f_k is enabled but not yet scheduled.

By fairness the $\bar{u}^j$'s are well defined, nonnegative and finite.

In addition they have the following properties:

Consider a transition $s_j \to^{f_k} s_{j+1}$, i.e., $i_{j+1} = k$. Then

(a) $u_k^j = 0$.

(b) For every $l \neq k$ such that $f_l(s_j) \neq \emptyset$, $u_l^j = u_l^{j+1} + 1$.

(c) For every $l \neq k$ such that $f_l(s_j) = \emptyset$, $u_l^j = u_l^{j+1}$.

Let σ now represent an infinite fair sequence. Assume that Q, W and ρ have been found satisfying the method's requirements. Consider the sequence of augmented states $(s_0, \bar{u}^0)$, $(s_1, \bar{u}^1), \ldots$.

By comparing properties (a), (b) and (c) to the requirements on Q and ρ, we obtain

$$\rho(s_0, \bar{u}^0) > \rho(s_1, \bar{u}^1) > \cdots,$$

contradicting the well foundedness of W. This shows that a successful choice of Q, W and ρ guarantees fair termination.

Conclusion: The delay variable method is sound.

Completeness is even more trivial. Assume that P is fairly convergent.

Take $Q = \mathrm{Acc}(I)$, i.e., true for all accessible states.

Take W to be $S \times N^m$, ρ the identity mapping

$$\rho(s, z_1, \ldots, z_m) = \langle s, z_1, \ldots, z_m \rangle.$$

The relation $>$ over W is defined as the transitive closure of the relation $>$ defined by

$$\langle s, z_1, \ldots, z_m \rangle > \langle s', z_1', \ldots, z_i', \ldots, z_m' \rangle \Leftrightarrow$$

$$\Leftrightarrow \exists i Q(s) \wedge s' \in f_i(s) \wedge \bigwedge_{j \neq i} [(f_j(s) \neq \emptyset \Rightarrow z_j = z_j' + 1)$$

$$\wedge (f_j(s) = \emptyset \Rightarrow z_j = z_j')] \wedge z_i = 0,$$

which holds exactly between two accessible augmented states that can appear contiguously in a computation. That this relation is well founded follows immediately from the fact that P is fairly convergent.

We conclude this section by an example of proving fair termination of the following distributed GCD program:

```
while $y_1 \neq y_2$ do if [$y_1 > y_2 \to y_1 := y_1 - y_2$ □ $y_1 < y_2 \to$ skip] fi od

                                   ||

while $y_1 \neq y_2$ do if [$y_1 > y_2 \to$ skip □ $y_1 < y_2 \to y_2 := y_2 - y_1$] fi od.
```

In our framework,

$$I = S = \{(y_1, y_2) \mid y_1 > 0, y_2 > 0\}.$$

The transition functions f_1, f_2 are given by

$$f_1([y_1, y_2]) = \textbf{if } y_1 > y_2 \textbf{ then } [y_1 - y_2, y_2] \textbf{ else if } y_1 < y_2 \textbf{ then } [y_1, y_2],$$

$$f_2([y_1, y_2]) = \textbf{if } y_1 > y_2 \textbf{ then } [y_1, y_2] \textbf{ else if } y_1 < y_2 \textbf{ then } [y_1, y_2 - y_1].$$

Note that both functions are disabled on states of the form $[y, y]$, which are therefore terminal.

To apply the delay variables method we choose Q as

$$Q([y_1, y_2]) = (y_1 \neq y_2) \wedge (y_1, y_2 > 0),$$

$W = N \times N$—the set of pairs of nonnegative integers with the lexicographic ordering on pairs given by

$$(m_1, n_1) > (m_2, n_2) \Leftrightarrow (m_1 > m_2) \text{ or } (m_1 = m_2 \text{ and } n_1 > n_2),$$

$$\rho([y_1, y_2], z_1, z_2) = (y_1 + y_2, \textbf{if } y_1 > y_2 \textbf{ then } z_1 \textbf{ else } z_2).$$

We have to show that the value of ρ decreases on each transition. Take for example the case of $y_1 > y_2$. We consider separately $i = 1$ and $i = 2$. We have to show

$$[y_1', y_2'] = f_1([y_1, y_2]) \wedge z_2 = z_2' + 1 \Rightarrow \rho([y_1, y_2], 0, z_2) > \rho([y_1', y_2'], z_1', z_2').$$

But certainly in this case $y_1 + y_2 > y_1' + y_2'$ so that $\rho > \rho'$. For $i = 2$ and $y_1 > y_2$ we have to show

$$[y_1', y_2'] = f_2([y_1, y_2]) \wedge z_1 = z_1' + 1 \Rightarrow \rho([y_1, y_2), z_1, 0) > \rho([y_1', y_2'), z_1', z_2').$$

But in this case $[y_1', y_2'] = [y_1, y_2]$ so that

$$\rho([y_1, y_2], z_1, 0) = (y_1 + y_2, z_1' + 1) > (y_1 + y_2, z_1') = \rho([y_1', y_2'], z_1', z_2').$$

This proves that the distributed GCD program is indeed fairly terminating.

3. Nondeterministic programs

In this section we develop the variant of the method appropriate to nondeterministic programs. The programs considered here will be presented in a structured language, and the method will lead to the establishment of *overall* fair total correctness.

The syntax of our programs is given by the following grammar:

$$\Pi ::= \textbf{skip} \mid x := t \mid \Pi ; \Pi \mid \textbf{if} \mathop{\square}_{i=1}^{m} B_i \rightarrow \Pi_i \textbf{ fi} \mid \textbf{while } B \textbf{ do } \Pi \textbf{ od}.$$

Here x is a program variable, t a term, B and B_i are boolean expressions. The booleans B_i in the context of the **if–fi** construct are called *guards*. Our language differs from that of Dijkstra [4] in that the loop is always guarded by a single condition.

3.1. Semantics

Let Var denote the set of program variables and $\mathcal{D}$ a domain of an interpretation. By a *state* we mean a mapping $s: \mathrm{Var} \to \mathcal{D}$.

Following [7] we define a simple operational semantics for programs based on a transition relation '$\to$' between configurations, that is pairs $\langle \Pi, s\rangle$ consisting of a program and a state. In addition we consider the two special states $\perp$ standing for divergence and **fail** standing for abortion.

In general $\langle \Pi, s\rangle \to \langle \Pi', s'\rangle$ means that one step of execution of Π applied to s can lead to a state s' with Π' being the remainder of Π yet to be executed.

It is convenient to assume the empty program E. Then Π' is E if Π terminates in s'. We assume that for any program Π, $E\,;\Pi = \Pi\,;E = \Pi$.

We define the above relation between configurations by the following clauses where $s \neq \mathbf{fail}$ and $s \neq \perp$:

(i) $\langle \mathrm{skip}, s\rangle \to \langle E, s\rangle$.

(ii) $\langle x := t, s\rangle \to \langle E, s'\rangle$, where $s'(x) = s(t)$ and $s'(y) = s(y)$ for $y \not\equiv x$.

(iii) $\langle \mathbf{if} \,\Box_{i=1}^{m}\, B_i \to \Pi_i \;\mathbf{fi}, s\rangle \to \langle \Pi_j, s\rangle$ if $\models B_j(s)$ $(1 \leq j \leq m)$.

(iv) $\langle \mathbf{if} \,\Box_{i=1}^{m}\, B_i \to \Pi_i \;\mathbf{fi}, s\rangle \to \langle E, \mathbf{fail}\rangle$ if $\models \bigwedge_{i=1}^{m} \neg B_i(s)$.

(v) $\langle \mathbf{while}\; B \;\mathbf{do}\; \Pi \;\mathbf{od}\; s\rangle \to \langle \Pi; \mathbf{while}\; B \;\mathrm{do}\; \Pi \;\mathbf{od}, s\rangle$ if $\models B(s)$.

(vi) $\langle \mathbf{while}\; B \;\mathbf{do}\; \Pi \;\mathbf{od}, s\rangle \to \langle E, s\rangle$ if $\models \neg B(s)$.

(vii) if $\langle \Pi, s\rangle \to \langle \Pi', s'\rangle$, then $\langle \Pi\,; \Pi_1, s\rangle \to \langle \Pi'\,; \Pi_1, s'\rangle$.

Let '$\to^*$' stand for the reflexive transitive closure of '$\to$'. We say that Π_0 *can diverge from* s_0 if there exists an infinite sequence

$$\langle \Pi_0, s_0\rangle \to \langle \Pi_1, s_1\rangle \to \cdots .$$

We say that Π *can fail from* s if for some Π_1

$$\langle \Pi, s\rangle \to^* \langle \Pi_1, \mathbf{fail}\rangle.$$

We may now define various semantics of programs by putting

$$\mathcal{M}_{\mathrm{p}}[\![\Pi]\!](s) = \{s' \mid \langle \Pi, s\rangle \to^* \langle E, s'\rangle\},$$

$$\mathcal{M}_{\mathrm{wt}}[\![\Pi]\!](s) = \mathcal{M}_{p}[\![\Pi]\!](s) \cup \{\perp \mid \Pi \text{ can diverge from } s\},$$

$$\mathcal{M}_{\mathrm{t}}[\![\Pi]\!](s) = \mathcal{M}_{\mathrm{wt}}[\![\Pi]\!](s) \cup \{\mathbf{fail} \mid \Pi \text{ can fail from } s\}.$$

We now proceed to define yet another semantics of programs—the one taking under consideration the assumption of fairness. The definition requires some way of distinguishing between various occurrences of the same subprogram.

A computation sequence

$$\sigma: \langle \Pi_0, s_0\rangle \to \langle \Pi_1, s_1\rangle \to \cdots ,$$

is said to be *fair* if it is either finite or for every program $\Pi : \mathbf{if} \,\Box_{i=1}^{m}\, B_i \to \Pi_i \;\mathbf{fi}\,; \Pi'$ and each $i = 1, \ldots, m$, if there are infinitely many j's for which $\langle \Pi, s_j\rangle$ appears in σ

and $\models B_i(s_j)$, then there are infinitely many j's among them such that the transition

$$\langle \Pi, s_j\rangle \to \langle \Pi_i ; \Pi', s_j\rangle$$

appears in σ.

This again captures the idea that every guard associated with a fixed location in the program, which is tested and found enabled an infinite number of times will be selected an infinite number of times.

To avoid confusion resulting from the fact that various occurrences of Π in σ do not need to correspond with the same program, we should actually label each subprogram of Π_0 with a unique label. It is clear how to perform this process and we leave it to the reader.

We may now define $\mathcal{M}_{\text{fair}}[\![\Pi]\!](s)$ analogously as $\mathcal{M}_{\text{t}}[\![\Pi]\!](s)$ by allowing $\perp$ in it only if Π can diverge from s by a fair computation sequence.

Let P, Q, R stand for formulae (assertions) in an assertion language which contains all program variables, terms and boolean expressions. We put $[P] = \{s \mid \models P(s)\}$. We stipulate that for any assertion P, $\perp \notin [P]$ and **fail** $\notin [P]$.

For any $f \in \{\text{p}, \text{wt}, \text{t}, \text{fair}\}$, assertions P, Q and a program Π we define

$$\mathcal{M}_f[\![\Pi]\!]([P]) = \bigcup_{s \in [P]} \mathcal{M}_f[\![\Pi]\!](s).$$

The statement of program correctness is defined by:

$$\models_f \{P\}\, \Pi\, \{Q\} \text{ iff } \mathcal{M}_f[\![\Pi]\!]([P]) \subseteq [Q].$$

We thus have four types of program correctness:

$\models_{\text{p}}$—partial correctness;

$\models_{\text{wt}}$—weak total correctness;

$\models_{\text{t}}$—total correctness;

$\models_{\text{fair}}$—total correctness under the assumption of fairness (fair total correctness).

The weak total correctness and the corresponding $\mathcal{M}_{\text{wt}}$ semantics are less often considered in the literature. We need these notions in the next section. We call the constructs $\{P\}\, \Pi\, \{Q\}$ the *correctness formulae.*

3.2. A transformation realizing fairness

In the subsequent considerations we need atomic programs of the form $x := ?$ called *random assignments.* $x := ?$ sets x to an arbitrary nonnegative integer. The semantics of random assignment is defined by adopting the clause

$$\langle x := ?, s\rangle \to \langle E, s'\rangle$$

for any state s' such that $s'(y) = s(y)$ for $y \not\equiv x$. We assume that x ranges over natural numbers which form a subset of the domain $\mathcal{D}$ of the interpretation.

Programs allowing random assignment have been extensively studied in [2]. In particular a system for proving total correctness of these programs has been presented there and we shall make use of it in order to develop proof rules for fair total correctness.

To this purpose we provide first a transformation of an arbitrary program Π into a program Π_{fair} allowing random assignments which realizes exactly all fair computations of Π. We proceed by the following successive steps:

Step 1. Replace each subprogram **if** $\square_{i=1}^{m} B_i \to \Pi_i$ **fi** of Π by the following subprogram:

$$\textbf{for}\, j := 1 \textbf{ to } m \textbf{ if } B_j \textbf{ then } z_j := z_j - 1;$$

$$\textbf{if} \mathop{\square}_{i=1}^{m} B_i \wedge z_i = 0 \wedge \bar{z} \geq 0 \to z_i := ?\,; \Pi_i \textbf{ fi},$$

where $\bar{z}$ stands for $z_1, \ldots, z_m$.

Step 2. Rename all variables $z_1, \ldots, z_m$ appropriately so that each **if–fi** construct has its 'own' set of these variables.

The variables $z_1, \ldots, z_m$ play here exactly the same role as in Section 2—they count down how many times the corresponding guard is enabled but not yet selected. The corresponding actions on these variables are incorporated in the program text.

The following lemma relates Π to Π_{fair}.

Lemma 3.1. *For any state* s,

$$\mathcal{M}_{\text{fair}}[\![\Pi]\!](s) = \mathcal{M}_{\text{wt}}[\![\Pi_{\text{fair}}]\!](s) \cup \{\textbf{fail}: \Pi \textit{ can fail from } s\}.$$

Here and later we disregard the problem that Π_{fair} can change the initial values of the (auxiliary) delay variables $z_1, \ldots,$ whereas Π cannot. It is easy to remedy this difficulty by retaining the initial values of these variables before the execution of Π_{fair} and restore them after the execution of Π_{fair}. We ignore this issue here since it is not relevant in the further discussion.

Proof. (a) We prove the $\subseteq$-inclusion. Let $\sigma = \langle \Pi_0, s_0 \rangle \to \langle \Pi_1, s_1 \rangle \to \cdots$ be a fair computation of Π. We extend it to a computation of Π_{fair} by assigning in each state of σ the values to the delay variables $z_i - s$. Given a state s_j there are two cases.

Case 1. For no state s_k $(k > j)$ the guard corresponding with z_j is selected.

Then by the assumption of fairness this guard is enabled only finitely many times in this computation. We put $s_j(z_i)$ to be equal $1 +$ the number of times this guard will be enabled beyond s_j.

Case 2. For some state s_k $(k > j)$ the guard corresponding with z_j is selected.

Then we put $s_j(z_i)$ to be equal $1 +$ the number of times this guard will be enabled before being next time selected.

(b) We prove the $\supseteq$-inclusion.

Let σ be a computation of Π_{fair}. Then its restriction to the computation steps dealing with Π is a computation sequence of Π. We show that it is a fair computation sequence. Suppose otherwise. Then the computation must be infinite and beyond some point in this computation some guard would be infinitely many times enabled and yet never chosen. By the construction of Π_{fair} the corresponding variable z_i would become arbitrarily small. This is however impossible because as soon as z_i becomes negative a failure will arise. □

Corollary 3.2. *Suppose that none of the delay variables occurs free in assertions P and Q. Then*

$$\models_{\text{fair}}\{P\}\,\Pi\,\{Q\} \textit{ iff } \forall s[\models P(s) \to \Pi \textit{ cannot fail from } s] \textit{ and } \models_{\text{wt}}\{P\}\,\Pi_{\text{fair}}\,\{Q\}.$$

3.3. A proof system for fair total correctness

Corollary 3.2 indicates that in order to prove fair total correctness of Π it is sufficient to prove weak total correctness of Π_{fair} provided the absence of failure in Π can be established.

To prove weak total correctness of Π_{fair} we can use the proof system introduced in [2], slightly modified for our purposes. The following axioms and proof rules are adopted.

(1) *Random assignment axiom*:

$$\{P\}\; x := ?\; \{P\},$$

provided x is not free in P.

(2) *Skip axiom*:

$$\{P\}\; \text{skip}\; \{P\}.$$

(3) *Assignment axiom*:

$$\{P[t/x]\}\; x := t\; \{P\},$$

where $P[t/x]$ stands for a substitution of t for all free occurrences of x in P.

(4) *Composition rule*:

$$\frac{\{P\}\,\Pi_1\,\{Q\},\{Q\}\,\Pi_2\,\{R\}}{\{P\}\,\Pi_1\,;\Pi_2\,\{R\}}$$

(5) *Selection rule*:

$$\frac{\{P \wedge B_i\}\,\Pi_i\,\{Q\}_{i=1,\dots,m}}{\{P\}\;\textbf{if}\;\square_{i=1}^{m}\,B_i \to \Pi_i\;\textbf{fi}\;\{Q\}},$$

(6) *While rule*:

$$\frac{\{P(\alpha) \wedge B\}\,\Pi\,\{\exists \beta < \alpha P(\beta)\}}{\{P(\alpha)\}\;\textbf{while}\;B\;\textbf{do}\;\Pi\;\textbf{od}\;\{\exists \beta \leqslant \alpha P(\beta) \wedge \neg B\}},$$

where α, β are variables ranging over ordinals (or more generally, well-founded sets).

(7) *Consequence rule*:

$$\frac{P \to P_1, \{P_1\}\, \Pi\, \{Q_1\}, Q_1 \to Q}{\{P\}\, \Pi\, \{Q\}}.$$

The above system is appropriate for proving weak total correctness of Π_{fair}. We call it WTC.

Consider now a proof of a correctness formula $\{P_1\}\, \Pi_{\text{fair}}\, \{Q_1\}$ in the above system. Due to the form of Π_{fair} this proof can be transformed into a proof of the correctness formula $\{P_1\}\, \Pi\, \{Q_1\}$ provided we use the following transformed version of the selection rule:

$$\frac{\{P\}\, \mathbf{if}_{\text{fair}}\, \mathbf{fi}\, \{Q\}}{\{P\}\, \mathbf{if}\, \square_{i=1}^{m}\, B_i \to \Pi_i\, \mathbf{fi}\, \{Q\}},$$

where $\mathbf{if}_{\text{fair}}\, \mathbf{fi}$ stands for the subprogram introduced in Step 1 of the transformation from Section 3.2.

The hypothesis of this rule can be simplified if we 'absorb' all assignments to delay variables into the assertion P and apply 'backwards' the original selection rule. In such a way we obtain a proof rule which deals exclusively with the **if**-construct and its components. It has the following form:

$$\frac{\{P[(B_j \to z_j + 1, z_j)/z_j]_{j \neq i}[1/z_i] \wedge B_i \wedge \bar{z} \geqslant 0\}\, \Pi_i\, \{Q\}_{i=1,\ldots,m}}{\{P\}\, \mathbf{if}\, \square_{i=1}^{m}\, B_i \to \Pi_i\, \mathbf{fi}\, \{Q\}},$$

where $B_j \to t_1, t_2$ stands for the conditional expression **if** B_j **then** t_1 **else** t_2 **fi**.

According to Corollary 3.2 we still have to deal with the issue of absence of failure. This problem can be taken care of in the usual way, i.e., by simply adding to the premises of the above rule the assertion

$$P \to \bigvee_{i=1}^{m} B_i.$$

Summarizing, the final version of the rule has the following form:

(8) *Fair selection rule*:

$$P \to \bigvee_{i=1}^{m} B_i,$$

$$\frac{\{P[(B_j \to z_j + 1, z_j)/z_j]_{j \neq i}[1/z_i] \wedge B_i \wedge \bar{z} \geqslant 0\}\, \Pi_i\, \{Q\}_{i=1,\ldots,m}}{\{P\}\, \mathbf{if}\, \square_{i=1}^{m}\, B_i \to \Pi_i\, \mathbf{fi}\, \{Q\}}.$$

We have thus obtained a proof system for proving fair total correctness of programs. It consists of the axioms (2), (3) and proof rules (4), (6)–(8). Note that the random assignment axiom is not needed—it was used only to derive the final form of the fair selection rule. Call this proof system FTC (for fair total correctness).

3.4. Soundness and completeness of FTC

Before we dwell on the issue of soundness and completeness of FTC we have to specify for which assertion languages and their interpretation is FTC an appropriate proof system.

We assume that the assertion language L contains two sorts: **data** and **ord**. We have a constant 0 of type **ord** and a binary predicate symbol $<$ over **ord**. Additionally, we assume that L includes second-order variables of arbitrary arity and sort. Thus we allow as atomic formulas, formulas of the form $(u_1, \ldots, u_n) \in a$. Such a formula is *well-formed* if the sorts and number of the variables $u_1, \ldots, u_n$ agree with the sort and arity of the second-order variable a.

Formulae are built up from atomic formulas by the usual boolean connectives and by quantification over variables of sorts **data** and **ord** and while set variables cannot be quantified over, they can be bound by the least fixed point operator μ.

We say that a set variable a always occurs *positively* in P if P is an a-positive formula. a-positive formulae are defined by induction as follows:

(i) if a does not occur free in P, then P is a-positive;

(ii) if $(u_1, \ldots, u_n) \in a$ is a well-formed formula, then it is a-positive;

(iii) if P, R are a-positive, then so are $\exists uP$, $\forall uP$, $P \wedge R$ and $P \vee R$;

(iv) if $P(a, b, u_1, \ldots, u_n)$ is both a-positive and b-positive and $(u_1, \ldots, u_n) \in b$ is a well-formed formula, then $\mu b(u_1, \ldots, u_n).P$ is a-positive.

Now, for any formula $P(a, u_1, \ldots, u_n)$ where $(u_1, \ldots, u_n) \in a$ is a well-formed atomic formula and a always occurs positively in P, the abstraction $\mu a(u_1, \ldots, u_n).P$ is also a formula of L. The free variables of $\mu a(u_1, \ldots, u_n).P$ are those of P other than a.

An interpretation J for this type of assertion language is an ordinary two-sorted second-order structure subject to the following four conditions:

(1) The domain $J_{\mathbf{data}}$ of sort **data** is countable and contains all natural numbers.

(2) The domain $J_{\mathbf{ord}}$ of sort **ord** is an initial segment of ordinals (to ensure a proper interpretation of the **while** rule).

(3) The constant 0 denotes the least ordinal and the predicate symbol $<$ denotes the strict ordering of the ordinals, restricted to $J_{\mathbf{ord}}$.

(4) The domains of each of the set sorts contain all sets of the appropriate kind (to ensure the existence of the fixed points considered below).

Let φ be a function assigning to each variable of L an element from the domain J of the appropriate type.

The truth under the interpretation J with respect to φ, written $\models_{J,\varphi}$, is defined in a standard way. The only nonstandard case is when a formula is of the form $\mu a(u_1, \ldots, u_n).P$. We put then $\models_{J,\varphi} \mu a(u_1, \ldots, u_n).P$ iff $\models_{J,\varphi[A/a]} P$, where $\varphi[A/a]$ is the modification of φ assigning set A to the variable a and where A is the least fixed point of the operator Φ defined by

$$\Phi(Z) = \{(i_1, \ldots, i_n) : \models_{J,\varphi[Z/a, i_1/u_1, \ldots, i_n/u_n]} P\}.$$

Since P is a-positive, the operator Φ is monotone and has a least fixed point.

Finally, we write $\models_J P$ iff for all $\varphi \models_{J,\varphi} P$ holds, i.e., if P is true with respect to J. The truth of the correctness formulas with respect to J is defined as before. We only need to indicate the dependence of the appropriate program semantics on the interpretation J.

By Tr_J denote the set of all formulae of L which are true with respect to J. Given a set of assertions AS and a proof system G for proving correctness formulae we denote by $\mathrm{AS} \vdash_G \varphi$ the fact that the correctness formula φ can be proved in G from the set of assumptions AS which can be used in the consequence rule.

After having introduced all these notions we can now state a lemma which is a proof theoretic counterpart of Corollary 3.2.

Lemma 3.3. *Suppose that none of the delay variables introduced in Π_{fair} occurs free in the assertions P and Q. Then for any interpretation J of the above kind,*

$$\mathrm{Tr}_J \vdash_{\mathrm{FTC}} \{P\}\,\Pi\,\{Q\} \;\textit{ iff }\; \mathrm{Tr}_J \vdash_{\mathrm{WTC}} \{P\}\,\Pi_{\text{fair}}\,\{Q\}$$
$$\textit{and } \forall s[\models_J P(s) \to \Pi \textit{ cannot fail from } s].$$

Proof. The proof is based on the analysis of the proofs in the corresponding proof systems and makes use of the Corollary 3.2. We leave the details to the reader. □

Corollary 3.2 and Lemma 3.3 reduce the question of soundness and completeness of the proof system FTC to that of WTC. But the results of [2] show that the proof system WTC is sound and complete for all interpretations J of the above kind. This shows that the proof system FTC is also sound and complete in the sense of the following theorem.

Theorem 3.4. *For all interpretations J of the above kind and all correctness formulae φ,*

$$\mathrm{Tr}_J \vdash_{\mathrm{FTC}} \varphi \;\textit{ iff }\; \models_{J,\text{fair}} \varphi.$$

3.5. An example of a proof in FTC

We conclude with an example, which can be dealt with using our system but not by any previous method. The program was suggested by Shmuel Katz. In this program we annotated each choice in a conditional statement by the name of the delay variable which is decremented if the choice is enabled (i.e., guard is true) but not selected. Let

```
Π:   while $x>0$ do
       if $\{z_1\}$ true → if $\{z_3\}$ $B \to x := x-1$
                           □ $\{z_4\}$ $B \to B :=$ false
                           □ $\{z_5\}\neg B \to$ skip        fi
       □ $\{z_2\}$ true → $B :=$ true
       fi
     od.
```

We want to prove

$$\models_{\text{fair}} \{\textbf{true}\}\ \Pi\ \{\textbf{true}\}, \tag{1}$$

i.e., that Π always terminates under the assumption of fairness.

The well-founded set we will consider is N^4 under lexicographic ordering. We have annotated each guard with an appropriate delay variable. There is a ranking function which underlies our formal proof which is given by

$$\rho(x, B, \bar{z}) = (x, z_3, 1-B, (B \to z_1, z_2)).$$

In the expression $1-B$, **true** is interpreted as 1, **false** as 0.

The crucial fact upon which the proof depends is that in a fair execution the value of ρ decreases on each iteration of the loop. We first demonstrate this fact informally providing the formal proof later. An iteration of the loop can be characterized by the guards which are selected or equivalently by the names of the delay variables associated with these guards.

Consider first the z_1, z_3 path. Here x is decremented so that ρ certainly decreases.

Along the z_1, z_4 path, the z_3 guard was enabled since B must have been true for z_4 to be selected. Consequently, z_3 is decremented, being an enabled but unselected guard. Since x remains the same, ρ again decreases. Along the z_1, z_5 path, B must have been false so that the fourth component of ρ is z_2 which is decremented when its guard is not selected.

In the z_2 path we have to distinguish between the case that B is initially false in which case $1-B$ drops from 1 to 0, and the case that B was initially true in which case the last component of ρ is z_1 which is decremented since z_2 is selected.

We now present a formal proof of (1). Let Π' be the body of the loop. We have to find an assertion $P(\alpha)$ such that

$$\{P(\alpha) \wedge \alpha > 0 \wedge x > 0\}\ \Pi'\ \{\exists \beta < \alpha P(\beta)\} \tag{2}$$

and

$$\exists \alpha P(\alpha). \tag{3}$$

We define

$$P(\alpha) \equiv x, \bar{z} \geqslant 0 \ \to\ \alpha = \rho(x, B, \bar{z}).$$

It is clear that (3) holds. To prove (2) we have to apply the fair selection rule so we have first to prove the premises

$$\{(P(\alpha) \wedge \alpha > 0 \wedge x > 0)[z_2 + 1/z_2][1/z_1] \wedge z_1, z_2 \geqslant 0\}\ \Pi_1\ \{\exists \beta < \alpha P(\beta)\} \tag{4}$$

and

$$\{(P(\alpha) \wedge \alpha > 0 \wedge x > 0)[z_1 + 1/z_1][1/z_2] \wedge z_1, z_2 \geqslant 0\}\ B := \textbf{true}\ \{\exists \beta < \alpha P(\beta)\} \tag{5}$$

as the first premise of the fair selection rule is obviously satisfied. Here

$$\begin{aligned}\Pi_1 \equiv \textbf{if } & B \to x := x-1\\ \square\ & B \to B := \textbf{false}\\ \square\ & \neg B \to \textbf{skip} \quad \textbf{fi}.\end{aligned}$$

To prove (4) we once again wish to apply the fair selection rule. The premises to prove are

$$\{P_1[B \to z_4+1, z_4/z_4][\neg B \to z_5+1, z_5/z_5][1/z_3] \wedge B \wedge z_3, z_4, z_5 \geqslant 0\}$$

$$x := x-1\ \{\exists \beta < \alpha P(\beta)\}, \tag{6}$$

$$\{P_1[B \to z_3+1, z_3/z_3][\neg B \to z_5+1, z_5/z_5][1/z_4] \wedge B \wedge z_3, z_4, z_5 \geqslant 0\}$$

$$B := \textbf{false}\ \{\exists \beta < \alpha P(\beta)\} \tag{7}$$

and

$$\{P_1[B \to z_i+1, z_i/z_i]_{i=3,4}[1/z_5] \wedge \neg B \wedge z_3, z_4, z_5 \geqslant 0\}\ \textbf{skip}\ \{\exists \beta < \alpha P(\beta)\}, \tag{8}$$

where $P_1 \equiv (P(\alpha) \wedge \alpha > 0 \wedge x > 0)[z_2+1/z_2][1/z_1] \wedge z_1, z_2 \geqslant 0$.

We have, by the assignment axiom,

$$\{\rho(x, 1, 0, 1) = \alpha \wedge B \wedge x > 0 \wedge \bar{z} \geqslant 0\}$$

$$x := x-1\ \{\rho(x+1, 1, 0, 1) = \alpha \wedge B \wedge x \geqslant 0 \wedge \bar{z} \geqslant 0\},$$

which implies by the consequence rule (6) as the necessary implications clearly hold.

To prove (7), note that the pre-assertion of (7) is equivalent to

$$\rho(x, z_3+1, 0, 1) = \alpha \wedge \alpha > 0 \wedge B \wedge \bar{z} \geqslant 0 \wedge x > 0,$$

which in turn implies the assertion

$$Q \equiv \exists \beta < \alpha\ (x > 0 \wedge \bar{z} \geqslant 0 \wedge \beta = \rho(x, z_3, 1, z_2)).$$

Now by the assignment axiom and the consequence rule

$$\{Q\}\ B := \textbf{false}\ \{\exists \beta < \alpha P(\beta)\},$$

so (7) by the consequence rule.

Finally, to prove (8) we note that

$$P_1[B \to z_i+1, z_i/z_i]_{i=3,4}[1/z_5] \wedge \neg B \wedge z_3, z_4, z_5 \geqslant 0$$

implies

$$\rho(x, z_3, 1, z_2+1) = \alpha \wedge \neg B \wedge \bar{z} \geqslant 0 \wedge x > 0$$

which in turn implies $\exists \beta < \alpha P(\beta)$. Hence (8) holds by the skip axiom.

Now, from (6)–(8) we get (4) by the fair selection rule.

To prove (5), note that the pre-assertion of (5) is equivalent to

$$\rho(x, z_3, 1-B, (B \to z_1+1, 1)) = \alpha \wedge \alpha > 0 \wedge x > 0 \wedge \bar{z} \geqslant 0,$$

which in turn implies the assertion

$$R \equiv \exists \beta < \alpha(\rho(x, z_3, 0, z_1+1) = \beta \wedge x > 0 \wedge \bar{z} \geq 0).$$

Now by the assignment axiom and the consequence rule $\{R\}$ $B :=$ **true** $\{\exists \beta < \alpha P(\beta)\}$ so (5) by the consequence rule.

We now have proved both (4) and (5) and we get (2) by the fair selection rule. Now (2) and (3) imply by the **while** rule $\{$**true**$\}$ Π $\{$**true**$\}$ so by virtue of the soundness of the system FTC we get (1). This concludes the proof.

4. On the size of needed ordinals

In the preceding sections we have presented methods for proving fair termination of (concurrent or structured nondeterministic) programs, using ranking functions into well-founded sets or predicates of ordinals. It is well known that any well-founded set $\langle W, > \rangle$ has an order preserving mapping into $\langle W_\alpha, > \rangle$ for some ordinal α, where $W_\alpha = \{\beta \mid \beta < \alpha\}$ (see [9] for details). Thus, one measure of the 'complexity' of fair termination of a concurrent program P is the least ordinal α for which there exist Q, W and ρ as in the delay variables method with $W = W_\alpha$. Let us call this ordinal α the 'fair ordinal' of P and denote it by α_P ($\alpha_P = 0$ in case P is not fairly convergent). A similar measure of complexity can be associated with structured nondeterministic programs by studying ordinals needed for applying the **while**-rule.

Consider bounds on α_P for natural classes of programs P. For definiteness we consider programs operating on natural numbers, i.e., the state space S is N^l for some l. In the case of concurrent programs each transition function corresponds to some recursive subset of N^l. (In fact, it suffices to look at transitions corresponding to assignments of the form $x := 0$, $x := y+1$, $x := y-1$ and guarded by tests of the form $x = 0$?, without affecting the following theorem.) Call such programs 'concurrent numerical programs'.

In the case of structured nondeterministic programs assume that all functions and relations used in the expressions are recursive (i.e., effectively calculable) and the usual functions and relations of Peano arithmetic are available in the language. Call such programs 'nondeterministic numerical programs'.

In the subsequent discussion we restrict our attention to nondeterministic numerical programs. Similar results can be proved for concurrent numerical programs. The complexity of fair termination of nondeterministic numerical programs is closely related to the complexity of numerical (nondeterministic) programs with random assignments.

The translation presented in Section 3.2 and the converse one replacing $x := ?$ by

```
x := 0; while B do if B → x := x + 1
                   □ B → B := false
                   fi
                od
```

show that both classes of programs are reducible to each other.

Since the proof rules for fair termination were obtained through the first translation, the ordinals α_P for both classes of programs are in fact the same. In [2] it was proved that exactly all recursive ordinals are needed to prove total correctness of numerical programs with random assignments. Hence the same result holds for the ordinals α_P associated with nondeterministic numerical programs.

We now prove the following stronger theorem concerning top level fairness only.

Theorem 4.1. *For any recursive ordinal α there exists a nondeterministic numerical program P with nondeterminism on a top level only, with α_P satisfying $\alpha_P \geq \alpha$.*

This theorem should be compared with [1], where the authors prove an analogous statement for $\alpha < \omega^\omega$ only.

Proof. We prove that each numerical program with random assignments which is otherwise deterministic is equivalent to a nondeterministic numerical program with top level fairness only. More precisely, we show that for each program Π of the first type there exists a nondeterministic numerical program Π_1 with nondeterminism on a top level only such that $\mathcal{M}_t[\![\Pi]\!] = \mathcal{M}_{\text{fair}}[\![\Pi_1]\!]$. The result then follows, since by [2] exactly all recursive ordinals are needed for proving total correctness of the programs of the first type.

Let Π be a program of the first type. Insert before each random assignment of the form $x := ?$ the assignment $x := 0$. By a well-known theorem Π is equivalent to a program Π' which contains one **while**-loop only and makes use of the auxiliary variable c ranging over labels attached to atomic programs and tests.

Assume that the labels form the set $\{1, \ldots, \text{halt}-1\}$ and that $\bar{x}$ is a vector of all variables of Π. Then we can assume that Π' is of the form

```
$c := 1$ ; $\bar{x} := \bar{t}$ ; while $c \neq \text{halt}$ do
                if $\mathop{\Box}_{i=1}^{\text{halt}-1}$ $c = i \rightarrow$ execute statement with label $i$; update $c$
                fi.
```

If the statement is a test then its execution is void but updating the counter c is performed accordingly to the value of the test. Replace now each part of the **if–fi** construct of the form $\Box\, c = i \rightarrow i : x := ?$; update c by $\Box\, c = i \rightarrow x := x + 1 \,\Box\, c = i \rightarrow$ update c. Call the resulting program Π_1.

By the construction the value of x just before updating the value of c to i is 0. It is now clear that Π_1 is the required program. □

5. Comparison with eventual-descent methods

Another approach to proving termination of concurrent programs is represented by the methods proposed in [6] and [9]. In this approach we do not require strict descent according to some well-founded measure at any step of the computation.

Instead we require that at any state some processors are guaranteed to cause a strict descent whenever they are activated. The identification of the 'helpful' processes is formulated in a way that ensures that by fairness eventually a strict descent will occur. Correspondingly we refer to the family of these methods as the methods of eventual descent.

Here we would like to compare the method of this paper, the delay variables method, with the method of eventual descent. For a convenient frame of comparison we will develop first the variant of the delay variables method that deals with justice (weak fairness).

Going back to the definition of execution sequences of concurrent programs, an execution sequence

$$\sigma : s_0 \rightarrow^{f_{i_1}} s_1 \rightarrow^{f_{i_2}} s_2 \rightarrow \cdots$$

is defined to be *just* if it is either finite, or if every transition f_k that is continuously enabled beyond some point in σ must be scheduled infinitely many times in σ.

Note the difference between justice and fairness. In just computations (execution sequences), a transition f_k is guaranteed to be scheduled infinitely many times only under the stronger requirement that f_k is *continuously* enabled beyond some point. Thus the notion of justice is weaker than that of fairness. Every fair computation is also just.

A program P is said to be *justly convergent* if every just computation of P is finite.

Following is the delay variables principle for proving that a program is justly convergent:

Find a state predicate $Q \subseteq S$, a well-founded set $(W, >)$ and a ranking function over extended states, $\rho : S \times N^m \rightarrow W$, such that:

(A1) $s \in I \Rightarrow (s \in T) \vee Q(s)$.

(A2) $Q(s) \wedge s' \in f_i(s) \Rightarrow (s' \in T) \vee Q(s')$ for $i = 1, \ldots, m$.

These two clauses ensure that the predicate Q holds for all accessible non-terminal states.

(A3) $Q(s) \wedge s' \in f_i(s) \wedge \bigwedge_{j \neq i} [f_j(s) \neq \emptyset \Rightarrow z_j = z_j' + 1] \Rightarrow$

$$\Rightarrow \rho(s, z_1, \ldots, z_{i-1}, 0, z_{i+1}, \ldots, z_m) > \rho(s', z_1', \ldots, z_m') \quad \text{for } i = 1, \ldots, m.$$

Note that the descent condition is even simpler for justice than it is for fairness. We decrement the jth delay variable whenever f_j is enabled but not activated, but allow it to be arbitrarily reset to a new value otherwise. In comparison, in the case of fairness, the delay variable z_j is required to retain its value in a transition from a state on which f_j is disabled. The result of this is that a transition which is disabled infinitely many times may never be activated. This of course is allowed in the case of justice as long as this transition is not *continuously* enabled beyond some point.

Showing the soundness of the delay principle for justice is very similar to the case of fairness. The only difference is in the definition of the particular delay values chosen for each computation. Given a computation σ we define: $u_k^j = (l-j)$, where l is the minimal index $l \geq j$ such that either f_k is disabled on s_l or f_k is the transition taken at s_l.

As an example of the application of the delayed justice principle we can take again the distributed GCD program that was used to illustrate fair termination, and prove that is also justly convergent. This can be done using the same Q and the same ρ. Hence the only additional check required is that ρ also satisfies clause (A3) in the justice principle, which is not too difficult to verify.

In order to compare the delay method with the methods proposed in [9] and [6], we present here the principle for proving just convergence from [9].

LPS-justice principle: Find a state predicate $Q \subseteq S$, a well-founded set $(W, >)$, a ranking function $\tilde{\rho}: S \to W$ and a *helpfulness* function $h: S \to [1, \ldots, m]$, such that:

(L1) $s \in I \Rightarrow (s \in T) \vee Q(s)$.

(L2) $Q(s) \wedge s' \in f_i(s) \Rightarrow (s' \in T) \vee Q(s')$ for $i = 1, \ldots, m$.

(L3) $Q(s) \wedge s' \in f_i(s) \Rightarrow [\tilde{\rho}(s) \geq \tilde{\rho}(s')]$ for $i = 1, \ldots, m$.

(L4) $Q(s) \wedge s' \in f_{h(s)}(s) \Rightarrow [\tilde{\rho}(s) > \tilde{\rho}(s')]$.

(L5) $Q(s) \wedge s' \in f_i(s) \wedge (\tilde{\rho}(s) = \tilde{\rho}(s')) \Rightarrow (h(s) = h(s'))$.

(L6) $Q(s) \Rightarrow f_{h(s)}(s) \neq \emptyset$.

Clauses (L1), (L2) are identical to clauses (A1), (A2) in the delay method. Clauses (L3), (L6) control the descent properties of the ranking function $\tilde{\rho}$. Clause (L3) ensures that the rank of a state never increases. Thus a strict descent is not guaranteed on each transition. However, as shown by (L4), the helpfulness function identifies for each states a transition $f_{h(s)}$, such that if $f_{h(s)}$ is activated on s, $\tilde{\rho}$ decreases strictly. Clause (L6) ensures that the helpful transition is always enabled. Clause (L5) states that any transition which is not a strictly decreasing transition preserves the identity of the helpful transition.

Let us refer to this principle as the LPS principle and to the delay variables principle for justice as the APS principle. Since both principles are complete, the only reasonable comparison between them is to ask whether a proof by one of them is easily (syntactically) reducible to a proof by the other. In that direction we have the following result.

Proposition 5.1. *The* APS *method is at least as easy to apply as the* LPS *method.*

Technically this implies that every proof by LPS is easily reducible to a proof by APS. Indeed, let Q, $(\tilde{W}, >)$, $\tilde{\rho}$ and h be the constructs found for the LPS method, shown to satisfy clauses (L1)–(L6) of the LPS method.

As the appropriate constructs for the APS method we choose Q to be the same, $W = \tilde{W} \times N$ with lexicographical ordering, and ρ defined by

$$\rho(s, \bar{z}) = (\tilde{\rho}(s), z_{h(s)}).$$

Let us show that these choices satisfy clauses (A1)–(A3) of the APS method.

Clauses (A1), (A2) are identical to (L1), (L2) and since the Q's are the same, must be satisfied. Consider clause (A3).

Let s satisfy Q and $s' \in f_i(s)$. Let $z_1, \ldots, z_m \in N^m$ and $z'_1, \ldots, z'_m \in N^m$ such that $z_i = 0$ and for every $j \neq i$ such that $f_j(s) \neq \emptyset$, $z'_j = z_j - 1$. There are two cases to consider. First, if $\tilde{\rho}(s) > \tilde{\rho}(s')$ then certainly

$$(\tilde{\rho}(s), z_{h(s)}) > (\tilde{\rho}(s'), z'_{h(s')}).$$

Due to (L3) the only other case is that $\tilde{\rho}(s) = \tilde{\rho}(s')$. Then, however, due to (L5) $h(s) = h(s')$. Let us denote $h(s) = h(s') = k$. In view of (L6), $f_k(s) \neq \emptyset$. Hence $z'_k = z_k - 1$ leading to $z_k > z'_k$. We thus have

$$(\tilde{\rho}(s), z_k) > (\tilde{\rho}(s'), z'_k)$$

also for this case.

We do not have at present any general result about a reduction in the other direction, i.e. given a $\rho : S \otimes N^k \to W$, find appropriate $\tilde{\rho} : S \to \tilde{W}$ and h. For all the cases we have considered, a choice of $\tilde{\rho}(s) = \rho(s, 0, \ldots, 0)$ seems to work, but we do not believe that this can be proven in general. Judging by the evidence that we do have we conclude by saying that the APS method suggested here is at least as good as the LPS method and possibly better.

References

[1] K.R. Apt and E.-R. Olderog, Proof rules dealing with fairness, in: *Workshop on Logic of Programs*, Lecture Notes in Computer Science **131** (Springer, Berlin, 1982) pp. 1–8; full-length version appeared in: *Sci. Comput. Programm.* **3** (1) (1983) 65–100.

[2] K.R. Apt and G.D. Plotkin, A cook's tour of countable nondeterminism, in: *Proc. 8th Colloquium on Automata Languages and Programming*, Acre 1981, Lecture Notes in Computer Science **115** (Springer, Berlin, 1981) pp. 477–493. Full version appeared as: Tech. Rept., Dept. of Computer Science, Edinburgh University, 1982.

[3] A. Chandra, Computable non-deterministic functions, in: *Proc. 19th Annual Symposium on Foundations of Computer Science* (1978) 127–131.

[4] E.W. Dijkstra, *A Discipline of Programming* (Prentice-H°ll, Englewood Cliffs, NJ, 1976).

[5] R.W. Floyd, Assigning meanings to programs, in: *Proc. AMS Symposium in Applied Mathematics* 19 (1967) 19–31.

[6] O. Grümberg, N. Francez, J.A. Makowsky and W.P. Roever, A proof rule for fair termination of guarded commands, in: J.W. de Bakker and J.C. van Vliet, eds., *Algorithmic Languages*, (North-Holland, Amsterdam, 1981) pp. 399–416.

[7] M.C.B. Hennessy and G.D. Plotkin, Full abstraction of a simple programming language, in: *Proc. 8th Symposium on Mathematical Foundations of Computer Science*, Lecture Notes in Computer Science **74** (Springer, Berlin, 1979) pp. 108–120.

[8] C.A.R. Hoare, An axiomatic basis of computer programming, *Comm. ACM* **12** (10) (1969) 576–580, 583.

[9] D. Lehmann, A. Pnueli and J. Stavi, Impartiality, justice and fairness: The ethics of concurrent termination, in: *Proc. 8th Colloquium on Automata Languages and Programming*, Lecture Notes in Computer Science **115** (Springer, Berlin, 1981) pp. 264–277.

[10] Z. Manna, *Mathematical Theory of Computation* (McGraw-Hill, New York, 1974).

[11] Z. Manna and A. Pnueli, Axiomatic approach to total correctness, *Acta Inform.* 3 (1974) 243–263.

[12] S. Owicki and D. Gries, An axiomatic proof technique for parallel programs, *Acta Inform.* **6** (1976) 319–339.