Science of Computer Programming 3 (1983) 65–100
North-Holland

PROOF RULES AND TRANSFORMATIONS DEALING WITH FAIRNESS*

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

E.-R. OLDEROG
Institut für Informatik und Praktische Mathematik, Christian-Albrechts-Universität Kiel, 2300 Kiel 1, Fed. Rep. Germany, and Oxford University Computing Laboratory, Programming Research Group, Oxford, OX1 3QD, United Kingdom

Communicated by A. Pnueli
Received April 1981
Revised April 1983

Abstract. We provide proof rules enabling the treatment of two fairness assumptions in the context of Dijkstra's do-od-programs. These proof rules are derived by considering a transformed version of the original program which uses random assignments $z := ?$ and admits only fair computations. Various, increasingly complicated, examples are discussed. In all cases reasonably simple proofs can be given. The proof rules use well-founded structures corresponding to infinite ordinals and deal with the original programs and not their translated versions.

1. Introduction

One of the troublesome issues concerning nondeterministic and parallel programs is *fairness*. Roughly speaking this assumption states that in the course of a computation every possible continuation is scheduled for execution sufficiently often. The meaning of a continuation depends on the programming language considered. For example, in the context of Dijkstra's guarded commands a possible continuation is a branch of the computation starting with a guard evaluating to true. *Sufficiently often* can be interpreted in a variety of ways the simplest being *eventually*.

The interest in fairness stems from the study of parallel programs where one wishes to express the fact that every component program will eventually finish the execution of each atomic instruction and start the next one. But since parallel programs can be translated into nondeterministic programs, also the notion of fairness has an easy translation. In fact, a number of authors found it convenient to study the phenomena of fairness (first) in the framework of nondeterministic programs—a practice we follow here in this paper.

* An extended abstract of a preliminary version of this paper appeared in [1].

The problem with the assumption of fairness is that it results in a noncontinuity of various semantic functions (see [2] for an overview of the literature on this subject). Consequently, various natural methods fail to work when applied here.

The aim of this paper is to develop a simple proof theoretic approach to the issue of fairness. This approach was originally suggested in Apt and Plotkin [2]. We restrict our attention here to the simplest framework in which fairness can be studied, namely Dijkstra's do-od-programs where the components are deterministic while-programs. Thus nondeterminism is allowed only at the top level. For these programs we introduce two fairness assumptions: *weak* and *strong* fairness.

The first step towards proof rules dealing with fairness is that for each fairness assumption we provide a transformation T yielding for a given do-od-program S an equivalent version $T(S)$ which uses *random assignments* of the form $z := ?$ (set z to an arbitrary nonnegative integer) to simulate exactly the fair computation of S. These transformations are interesting in their own right, but in this paper we use them only as a preparation for the second step. In this step the Hoare-style proof rules of Apt and Plotkin [2], which can cope with random assignments, are applied to $T(S)$ in order to derive in a systematic way new proof rules for S dealing with fairness. It should be stressed that these proof rules deal with the *original* program S—the transformation T is absorbed, as it were, into the assertions and the proof rules leaving the program S intact. Our proof rules use *well-founded structures corresponding to infinite ordinals* to prove total correctness under the assumptions of weak or strong fairness, respectively.

The use of such infinitistic methods seems to be necessary in view of the results by Emerson and Clarke [6] who showed that termination under fairness assumption is not first order definable. This result prompted them to formulate a conjecture that no useful sound and relatively complete proof system for this property exists. We hope that our results show that one still can reason about fairness in a simple and natural way by resorting to infinitistic means. Moreover, the results in [2] imply soundness and relative completeness of our system for a special type of assertion language—one which allows the use of the least fixed point operator and ordinals. Thus our system disproves Emerson and Clarke's conjecture.

This paper is organized as follows. The next section defines the notions of weak and strong fairness for programs of the form $S = \mathbf{do}\ B_1 \to S_1 \ldots B_n \to S_n\ \mathbf{od}$. If the guards $B_1, \ldots, B_n$ are identical, or more generally $B_1 \leftrightarrow B_2 \leftrightarrow \cdots \leftrightarrow B_n$ is a loop invariant, weak and strong fairnesss coincide and we simply talk of fairness. In Section 3 a proof rule dealing with fairness is developed, first for the case of two identical guards $B_1 = B_2$. To illustrate the proposed proof method we study three, increasingly more complicated, examples in Section 4. Next, in Section 5, the proof rule for fairness is extended to cover the case of n identical guards $B_1 = \cdots = B_n$. Three examples for the extended proof rule are discussed in Section 6. The general case of n arbitrary guards $B_1, \ldots, B_n$ is analyzed in Section 7. In this case it is necessary to distinguish between weak and strong fairness. The differences are reflected both in the transformations and the derived proof rules. Section 7 contains

also a detailed correctness proof for the considered program transformations. In Section 8 three examples concerning strong fairness, related to those in Section 4, are studied. In the following sections two natural examples where program correctness relies on the assumption of fairness are investigated—in Section 9 the problem of zero searching is treated and in Section 10 the example of asynchronous fixed point computation. Section 11 investigates which ordinals are needed for the proofs. We show that at least all ordinals $\alpha < \omega \cdot \omega^{\omega}$ are necessary. (This result was in the meantime greatly improved in [3].) Finally, in Section 12 we assess our approach and relate it to several other approaches such as [10] and [11].

2. Definitions

We consider nondeterministic do-od-programs of the form

$$S \equiv \textbf{do } B_1 \to S_1 \,\square \cdots \square\, B_n \to S_n \textbf{ od}$$

with n subprograms S_i guarded by Boolean expressions B_i, $i = 1, \ldots, n$. Throughout this paper we assume that the subprograms S_i are simple deterministic while-programs so that only one level of nondeterminism—and hence only one level of fairness—is studied. (The discussion of how our approach generalizes to the case of nested nondeterminism is postponed to Section 12.)

The program S manipulates *states*, i.e. mappings $\sigma : Var \to \mathcal{D}$. Var is a fixed set of variables which covers the program variables of S, but may also include additional variables if necessary. $\mathcal{D}$ is the domain of the underlying interpretation J. Variables $v \in Var$ may be restricted to range only over a subset $\mathcal{D}_v \subseteq \mathcal{D}$. The symbol $\models$ is used to denote *validity* or *truth* of Boolean expressions (and later on also of assertions and correctness formulas) under J. Thus $\models B(\sigma)$ states that the Boolean expression B is true in state σ. The meaning of a subprogram S_i of S is simply a partially defined mapping $\mathcal{M}(S_i)$ from states to states. To define notions of fairness and total correctness we introduce the concept of computation sequences. For $i \in \{1, \ldots, n\}$ and states σ, σ' we write

$$\sigma \xrightarrow{i} \sigma' \text{ if } \models B_i(\sigma) \quad \text{and} \quad \mathcal{M}(S_i)(\sigma) = \sigma'$$

and

$$\sigma \xrightarrow{i} \perp \text{ if } \models B_i(\sigma) \quad \text{and} \quad \mathcal{M}(S_i)(\sigma) \text{ is undefined, i.e. } S_i \text{ diverges from } \sigma.$$

Note that the restriction to one level of nondeterminism in S leads us to view the execution of S_i as a single step in the computation of S.

Computation sequences of S are now those sequences ξ of states which belong to one of the following categories:

(1) $\quad \xi = \sigma_1 \xrightarrow{i_1} \sigma_2 \xrightarrow{i_2} \cdots \xrightarrow{i_{m-1}} \sigma_m$

where $i_j \in \{1, \ldots, n\}$ and $\models (\neg B_1 \wedge \cdots \wedge \neg B_n)(\sigma_m)$. Then ξ is said to *terminate*.

(2) $\xi = \sigma_1 \xrightarrow{i_1} \sigma_2 \xrightarrow{i_2} \cdots \xrightarrow{i_{m-1}} \perp$

where $i_j \in \{1, \ldots, n\}$. Then ξ is said to *fail*.

(3) $\xi = \sigma_1 \xrightarrow{i_1} \sigma_2 \xrightarrow{i_2} \cdots \xrightarrow{i_{j-1}} \sigma_j \xrightarrow{i_j} \cdots$

where $i_j \in \{1, \ldots, n\}$ and ξ is infinite. Then ξ is said *to diverge*. A computation sequence ξ of S is called *weakly fair* if ξ is either of the form (1) or (2), or of the form (3), but then satisfies the following condition:

$$\forall i \in \{1, \ldots, n\} \left(\left(\overset{\infty}{\forall} j \in \mathbb{N} \models B_i(\sigma_j) \right) \rightarrow \left(\overset{\infty}{\exists} j \in \mathbb{N}\, i_j = i \right) \right).$$

The quantifier $\overset{\infty}{\forall}$ means "for all, but finitely many" and $\overset{\infty}{\exists}$ stands for "there exist infinitely many". Thus the above condition states that if B_i is almost always true then the ith subprogram S_i is infinitely often chosen. (This notion of weak fairness is called 'justice' in [11].)

A computation sequence ξ of S is called *strongly fair* if ξ is either of the form (1) or (2) or of the form (3), but then satisfies the following condition:

$$\forall i \in \{1, \ldots, n\} \left(\left(\overset{\infty}{\exists} j \in \mathbb{N} \models B_i(\sigma_j) \right) \rightarrow \left(\overset{\infty}{\exists} j \in \mathbb{N}\, i_j = i \right) \right).$$

In words: if B_i is infinitely often true then the ith subprogram S_i is infinitely often chosen. (This notion of strong fairness corresponds to the notion of 'fairness' in [11].)

Note that if $B_1 = \cdots = B_n$ holds, or more generally $B_1 \leftrightarrow B_2 \leftrightarrow \cdots \leftrightarrow B_n$ is a loop invariant, then weak and strong fairness coincide.

In this case we shall simply talk of *fairness*. (This concept of fairness corresponds to a special case of the notion of 'impartiality' in [11].)

To distinguish between total correctness of programs S under (1) *weak fairness assumption*, (2) *strong fairness assumption*, (3) *fairness assumption*, (4) *no further assumption*, we introduce four kinds of correctness formulas:

(1) $weak \rightarrow \{P\}\, S\, \{Q\}$,

(2) $strong \rightarrow \{P\}\, S\, \{Q\}$,

(3) $fair \rightarrow \{P\}\, S\, \{Q\}$,

(4) $\{P\}\, S\, \{Q\}$,

where P and Q are assertions, i.e. formulas allowing quantifiers. The *validity* of these correctness formulas is defined as follows:

(1) $\models weak \rightarrow \{P\}\, S\, \{Q\}$ holds if every *weakly fair* computation sequence of S starting in a state σ satisfying $\models P(\sigma)$ terminates in a state satisfying Q, i.e. is of the form $\sigma \rightarrow^{i_1} \cdots \rightarrow^{i_m} \sigma'$ where $\models Q(\sigma')$ holds for the final state σ'.

(2) $\models strong \rightarrow \{P\}\, S\, \{Q\}$: replace *weakly fair* by *strongly fair* in (1).

(3) $\models fair \to \{P\}\, S\, \{Q\}$ holds if both $\models weak \to \{P\}\, S\, \{Q\}$ and $\models strong \to \{P\}\, S\, \{Q\}$. In the following this notation will be used only if $B_1 = \cdots = B_n$ holds in S.

(4) $\models\{P\}\, S\, \{Q\}$: delete *weakly fair* in (1), i.e. the requirement of (1) should hold for *every* computation.

Thus under fairness assumptions we need not bother about unfair computation sequences. We shall soon study examples of programs which behave properly only under fairness assumptions. But first let us develop Hoare-style proof rules which by themselves prevent us from attempting to prove something about unfair computation sequences.

3. Fairness: 2 guards

We start with the development of a proof rule for total correctness under (*weak* = *strong*) fairness assumption for programs

$$S \equiv \textbf{do}\ B \to S_1 \,\square\, B \to S_2\ \textbf{od}$$

with only two subprograms guarded by identical Boolean expressions. The first step in this development is the simulation of fairness using random assignments of the form $z := ?$ which assign an arbitrary nonnegative integer value to the variable z. (We remark that random assignments lead to unbounded, but countable nondeterminism. Semantics and proof theory for programs allowing this kind of nondeterminism have been studied in [2].)

To achieve this simulation we transform S into the following program:

$$\begin{aligned} T^2_{fair}(S) \equiv\ & z_1 := ?;\, z_2 := ?; \\ & \textbf{do}\ B \wedge z_1 \leqslant z_2 \to S_1;\, z_1 := ?;\, z_2 := z_2 - 1 \\ & \square\, B \wedge z_2 < z_1 \to S_2;\, z_1 := z_1 - 1;\, z_2 := ? \\ & \textbf{od}. \end{aligned}$$

Here z_1 and z_2 are new variables not occurring in S which range over the integers. These variables are added to the program S in order to implement a scheduler in $T^2_{fair}(S)$ which decides which of the subprograms S_1 and S_2 is to be executed next.

This is done as follows: at every moment in a computation of $T^2_{fair}(S)$ the values of the variables z_1 and z_2 represent the *priorities* assigned to the subprograms S_1 and S_2. Consequently z_1 and z_2 will be called *priority variables*. We say that S_1 has a higher priority than S_2 if $z_1 < z_2$ holds (and vice versa for $z_2 < z_1$). The guards "$B \wedge z_1 \leqslant z_2$" and "$B \wedge z_2 < z_1$" guarantee that the subprogram with the higher priority is scheduled for execution. If both subprogram have the same priority, S_1 gets executed. After every execution of a subprogram, say S_i, the priority of the other, not chosen subprogram, say S_j, gets *increased* (by decrementing z_j by 1) whereas the priority of S_i is reset to some arbitrary nonnegative value. Gradually

increasing the priority of S_j excludes the possibility of executing S_i forever. This guarantees fairness. At the very beginning of a computation of $T^2_{fair}(S)$ both priority variables get arbitrary nonnegative values. The value of a priority variable z_j plus 1 describes also the maximum number of computation steps which may pass *before* the subprogram S_j is eventually scheduled for execution. Therefore the variables z_1, z_2 may also be called *counter variables* or (following [3]) *delay variables*. These, however are only informal explanations showing that $T^2_{fair}(S)$ allows only *fair* computations. The precise relationship between S and $T^2_{fair}(S)$ is stated in

Lemma 1. (i) *If ξ is a fair computation sequence of S then there exists an extension ξ' of ξ including the new variables z_1 and z_2 such that ξ' is a computation sequence of $T^2_{fair}(S)$.*

(ii) *Conversely, if ξ' is a computation sequence of $T^2_{fair}(S)$ then its restriction ξ to the variables of S is a fair computation sequence of S.*

For computation sequences $\xi = \sigma_1 \rightarrow^{i_1} \cdots \sigma_j \rightarrow^{i_j} \cdots$ and $\xi' = \sigma'_1 \rightarrow^{i_1} \cdots \sigma'_j \rightarrow^{i_j} \cdots$ we say that ξ' is an *extension of ξ to the variables $z_1, \ldots, z_n$* if the states σ'_j differ from σ_j at most in the variables $z_1, \ldots, z_n$. And ξ is called *a restriction of ξ' to the variables $x_1, \ldots, x_m$* if every state σ_j is obtained from σ'_j by resetting every variable $z \notin \{x_1, \ldots, x_m\}$ to its value in σ'_1, i.e. by defining

$$\sigma_j(z) = \begin{cases} \sigma'_j(z) & \text{if } z \in \{x_1, \ldots, x_m\}, \\ \sigma'_1(z) & \text{otherwise.} \end{cases}$$

Lemma 1 states that T^2_{fair} is a *faithful transformation* in the sense that for every program S of the form considered here $T^2_{fair}(S)$ simulates *exactly* all fair computations sequences of S.

Proof of Lemma 1. (i) Consider a fair computation sequence

$$\xi = \sigma_1 \xrightarrow{i_1} \cdots \sigma_j \xrightarrow{i_j} \cdots$$

of S with $i_j \in \{1, 2\}$. We explain now how to extend ξ to a sequence

$$\xi' = \sigma'_1 \xrightarrow{i_1} \cdots \sigma'_j \xrightarrow{i_j} \cdots$$

by providing new values to the variables z_1 and z_2. For $l \in \{1, 2\}$ we define

$$\sigma'_j(z_l) = min\,\{k - j \mid k \geqslant j \wedge (i_k = l \vee \models \neg B(\sigma_k))\}.$$

Note that this minimum always exists because ξ is fair. By construction in every state σ'_j (except the final state σ'_k with $\models \neg B(\sigma'_k)$ if ξ is finite) exactly one of the variables z_1 and z_2 has the value 0, namely z_{i_j}. The other variable has a positive value. Thus the scheduler built into the program $T^2_{fair}(S)$ would indeed choose the subprogram S_{i_j} when started in the state σ'_j. Further on, the values of the variables z_1 and z_2 in the states σ'_j have been defined in a way which is consistent with the assignments to these variables in $T^2_{fair}(S)$. This shows that ξ' is in fact a valid computation sequence of $T^2_{fair}(S)$.

(ii) Let ξ' be a computation sequence of $T^2_{fair}(S)$ and let ξ be its restriction to the variables of S. It is obvious that ξ is a valid computation sequence of S. But we have to prove that this computation sequence is fair.

Suppose this is not the case. Then ξ is infinite, i.e. B is always true, but one subprogram of S, say S_i, is never scheduled for execution from a certain moment on. By the definition of $T^2_{fair}(S)$, the value of the variable z_i becomes arbitrarily small and from some moment on smaller than -1. But this is impossible because it is easy to check that in every state σ'_j of ξ' the following invariant:

$$IN \equiv z_1 \geqslant -1 \wedge z_2 \geqslant -1$$
$$\wedge (z_1 = -1 \rightarrow z_2 \geqslant 0) \wedge (z_2 = -1 \rightarrow z_1 \geqslant 0)$$

holds. □

Though lemma 1 is interesting in its own right, we shall use here only a corollary stated in

Proposition 1. *Let P and Q be assertions not having z_1 or z_2 as free variables. Then*

$$\models fair \rightarrow \{P\}\, S\, \{Q\} \text{ iff } \models \{P\} T^2_{fair}(S) \{Q\}.$$

Thus in order to prove total correctness of S under fairness assumption it suffices to prove total correctness of $T^2_{fair}(S)$ in the usual sense. But we do not recommend the actual transformation of S into $T^2_{fair}(S)$ as part of a proof method. This approach would correspond to Flon and Suzuki's idea to employ certain transformations as proof rules in a system for total correctness of parallel programs [8], with the disadvantage of destroying the structure of the original program S.

Instead—in a second step—we use the transformation conceptually in order to derive a direct proof rule for S dealing with fairness. This derivation applies the proof methods of Apt and Plotkin [2] to $T^2_{fair}(S)$ and then reads back the ensuing rule in terms of the original program S. Thus in order to apply the derived proof rule one never has to carry out the transformations of S into $T^2_{fair}(S)$ explicitly.

Let us present now this step. Let $(W, >)$ be a *well-founded structure*, i.e. $>$ is a partial order relation over the set W such that there is no infinite descending chain

$$w_1 > w_2 > w_3 > \cdots$$

in W. And let α, β, γ be variables ranging over W. Then the relevant proof rule of [2] for do-od-programs like $T^2_{fair}(S)$ is

$$(*) \quad \frac{\begin{array}{l} P \rightarrow \exists \alpha\, \tilde{R}(\alpha) \\ (\exists \alpha\, \tilde{R}(\alpha) \wedge \neg(\tilde{B}_1 \vee \cdots \vee \tilde{B}_n)) \rightarrow Q \\ i = 1, \ldots, n: \\ \{\tilde{R}(\alpha) \wedge \tilde{B}_i\} \tilde{S}_i \{\exists \beta < \alpha\, \tilde{R}(\beta)\} \end{array}}{\{P\}\ \textbf{do}\ \tilde{B}_1 \rightarrow \tilde{S}_1 \,\square \cdots \square\, \tilde{B}_n \rightarrow \tilde{S}_n\ \textbf{od}\ \{Q\}}$$

Rule (*) formalizes Floyd's principle [9] stating that in order to prove total correctness of a loop, one has to find an appropriate loop invariant $\tilde{R}$ containing a variable α ranging over well-founded structure W which decreases with every loop execution. The extension contained in (*) is that the subprograms $\tilde{S}_i$ may be nondeterministic and even allow random assigments. As a consequence one cannot prove total correctness with natural numbers $(\mathbb{N}_0, >)$ any more, but has to resort to infinitistic methods in form of general well-founded structures $(W, >)$. This fact is proved in Apt and Plotkin [2].

We remark that [2] uses ordinals instead of well-founded structures, but both approaches are equivalent: every well-founded structure $(W, >_w)$ can be embedded into a well-founded structure of the special form $(W_\alpha, >)$ where α is an ordinal, $>$ the natural order for ordinals, and $W_\alpha = \{\beta \mid \alpha > \beta\}$ the set of all ordinals β smaller than α (cf. [11]). Nevertheless, it is sometimes easier to work directly with well-founded structures instead of with their representations as ordinals (see Section 10).

In addition we need an axiom covering the random assignments $z := ?$ inside $T^2_{fair}(S)$. This is

(**) $\{\forall z\, P\}\ z := ?\ \{P\}$

Applying now (*) and (**) to $T^2_{fair}(S)$ yields the following:

($FAIR_2$) *Proof Rule for Fairness*: 2 *guards*:

(1) $P \to \forall z_1, z_2\, \exists \alpha\, R(\alpha, z_1, z_2)$

(2) $\exists \alpha\, R^{IN}(\alpha, z_1, z_2) \wedge \neg B \to Q$

(3) $\{R^{IN}(\alpha, z_1, z_2) \wedge B \wedge turn = 1\}$
S_1
$\{\forall z_1\, \exists \beta < \alpha\, R^{IN}(\beta, z_1, z_2 - 1)\}$

(4) $\{R^{IN}(\alpha, z_1, z_2) \wedge B \wedge turn = 2\}$
S_2
$\{\forall z_2\, \exists \beta < \alpha\, R^{IN}(\beta, z_1 - 1, z_2)\}$

(5) $fair \to \{P\}\ \mathbf{do}\ B \to S_1 \,\square\, B \to S_2\ \mathbf{od}\ \{Q\}$

where z_1, z_2 are new variables not occurring in P, S, Q. The notation $R(\alpha, z_1, z_2)$ indicates the variables α, z_1, z_2 may occur freely in the assertion R. This is convenient for denoting substitutions implicitly. For example, $R(\beta, z_1, z_2 - 1)$ stands for R with β and $z_2 - 1$ substituted for α and z_2, respectively.

The expression $turn = 1$ abbreviates $z_1 \leqslant z_2$ and $turn = 2$ abbreviates $z_2 < z_1$. The assertion $R^{IN}(\alpha, z_1, z_2)$ is defined by

$$R^{IN}(\alpha, z_1, z_2) \equiv R(\alpha, z_1, z_2) \wedge IN$$

where IN is the assertion

$$IN \equiv z_1 \geqslant -1 \wedge z_2 \geqslant -1$$
$$\wedge (z_1 = -1 \rightarrow z_2 \geqslant 0) \wedge (z_2 = -1 \rightarrow z_1 \geqslant 0)$$

which is a standard invariant of T^2_{fair} due to Lemma 1.

The proof rule $FAIR_2$ is *sound* in the sense that whenever all premises (1)–(4) are valid then also the conclusion (5) is valid. It is also *relatively complete* in the sense that whenever the conclusion (5) is valid we can find an appropriate well-founded structure $(W, >)$ and an invariant $R(\alpha, z_1, z_2)$ such that the premises (1)–(4) are valid. It is an advantage of our transformation technique that these results are immediate consequences of Proposition 1 together with the soundness and relative completeness of rule (*) as proved in [2]. We remark that according to [2] the invariant $R(\alpha, z_1, z_2)$ can be expressed in an assertion language allowing a least fixed-point operator μ besides variables ranging over W and the usual first order quantifiers.

Clearly introducing the abbreviation '*turn*' and the additional invariant IN is needed neither for soundness nor relative completeness of the proof rule $FAIR_2$. But these additions are convenient when applying the rule to examples. The standard invariant IN can simply be assumed while formulating the invariant $R(\alpha, z_1, z_2)$ and '*turn*' improves the readability of $R(\alpha, z_1, z_2)$.

In general, finding an appropriate invariant $R(\alpha, z_1, z_2)$ means estimating the number α of do-od-loop executions of the program S. This estimation can be expressed in terms of the program variables in S with additional help of the priority (or delay) variables z_1 and z_2. Fortunately, in the case of only two guards we can often represent this number α of loop executions as a (lexicographically ordered) pair

$$\alpha = \langle \beta, z \rangle$$

where β counts the number of rounds the program S will still execute before termination and z counts the number of computation steps in the current round. By a *round* we mean here a sequence of computation steps with maximal length for which always subprogram S_1 or always subprogram S_2 is executed. Partitioning α into β and z is very helpful in those cases where the number of rounds β can be estimated *independently* of their lengths z. To do this we shall use '*turn*' as an additional variable, and not just as an abbreviation for $z_1 \leqslant z_2$ resp. $z_2 < z_1$.

This informal discussion is made precise in the following:

(*S-FAIR*$_2$) *Simplified Proof Rule for Fairness*: 2 *guards*:

(1) $P \rightarrow \forall\, turn\ \exists \beta\, R^*(\beta, turn)$

(2) $\exists\beta\, R^*(\beta, turn) \wedge \neg B \to Q$

(3) $\{R^*(\beta, 1) \wedge B\}$

S_1

$\{(\exists\gamma \leq \beta\, R^*(\gamma, 1)) \wedge \exists\gamma < \beta\, R^*(\gamma, 2)\}$

(4) $\{R(\beta, 2) \wedge B\}$

S_2

$\{(\exists\gamma \leq \beta\, R^*(\gamma, 2)) \wedge \exists\gamma < \beta\, R^*(\gamma, 1)\}$

---

(5) $fair \to \{P\}$ **do** $B \to S_1 \,\square\, B \to S_2$ **od** $\{Q\}$

where *turn* is a new variable not occurring in P, S, Q which ranges over the set $\{1, 2\}$

Note that in the premise (3) and (4) the value of α is decreased only if a switch of control to the other subprogram occurs. This formalizes the intuitive idea that α now counts the number of rounds instead of the numbers of loop executions. Let us now prove the *soundness* of the simplified rule *S-FAIR*$_2$ by proving the following:

Lemma 2. *If there is an invariant $R^*(\beta, turn)$ satisfying the premises of the simplified proof rule S-FAIR$_2$ then there exists also an invariant $R(\alpha, z_1, z_2)$ such that $R^{IN}(\alpha, z_1, z_2)$ satisfies the premises of the original rule FAIR$_2$.*

Proof. Let $(W, >)$ be the well-founded structure belonging to $R^*(\beta, turn)$. Then $R(\alpha, z_1, z_2)$ can be constructed from $R^*(\beta, turn)$ as follows:

$$\begin{aligned} R(\alpha, z_1, z_2) \equiv\ & (B \wedge z_1 \leq z_2 \to \exists\beta : \alpha = \langle\beta, z_2\rangle \wedge R^*(\beta, 1)) \\ & \wedge (B \wedge z_2 < z_1 \to \exists\beta : \alpha = \langle\beta, z_1\rangle \wedge R^*(\beta, 2)) \\ & \wedge (\neg B \to \exists\beta, turn, z : \alpha = \langle\beta, z\rangle \wedge R^*(\beta, turn)) \end{aligned}$$

where pairs $\langle\beta, z\rangle$ are ordered lexicographically:

$$\langle\beta_1, z_1\rangle > \langle\beta_2, z_2\rangle \text{ if } \beta_1 > \beta_2 \vee (\beta_1 = \beta_2 \wedge z_1 > z_2).$$

Note that by the standard invariant *IN* we know that always $z_1 \geq -1 \wedge z_2 \geq -$ holds. Thus the pairs $\langle\beta, z\rangle$ to be considered in $R^{IN}(\alpha, z_1, z_2)$ are all elements of $W \times \{z \mid z \geq -1\}$ which is indeed a well-founded structure under lexicographical order. It is now easy to check that $R^{IN}(\alpha, z_1, z_2)$ satisfies the premises (1)–(4) of rule *FAIR*$_2$. We remark that *IN* is crucial for verifying the premises (3) and (4). □

We remark that we have not been able to prove relative completeness of the simplified rule *S-FAIR*$_2$, i.e. the converse of Lemma 2. The stumbling block is that we have no counter variable z at our disposal. Nevertheless, *S-FAIR*$_2$ is a ver

useful, sound proof rule which is convenient to apply as we shall see in the next section.

4. Examples for fairness: 2 guards

We now treat a small hierarchy of examples with our proof rules for fairness in order to demonstrate that the invariants $R(\alpha, z_1, z_2)$ and $R^*(\beta, turn)$ can indeed be chosen in such a way that they just formalize the intuitive meaning of α and β, respectively. All examples presented here deal with termination only, i.e. with correctness formulas of the form $fair \rightarrow \{\mathbf{true}\}\, S\, \{\mathbf{true}\}$. Letters $A, B, \ldots$ denote Boolean variables and letters $x, y, \ldots$ integer variables.

Example 1.

$$\vDash fair \rightarrow \{\mathbf{true}\}\ \mathbf{do}\ A \rightarrow skip \,\square\, A \rightarrow A := \mathbf{false}\ \mathbf{od}\ \{\mathbf{true}\}.$$

This is essentially the program studied in Dijkstra [5, p. 76]. Note that this termination result does not hold without fairness assumption. Let us first apply the original rule $FAIR_2$ where we need an invariant $R(\alpha, z_1, z_2)$. As we know α is intended to count the numbers of *loop executions*. To determine α let us analyze the possible cases. First, if A is false at the beginning the program terminates immediately so that $\alpha = 0$. Otherwise we know that $\alpha > 0$. Suppose we start with the second subprogram $S_2 \equiv A := \mathbf{false}$. Then simply $\alpha = 1$. The more interesting case is when we start with the first subprogram $S_1 \equiv$ skip. Now we cannot predict the exact number α of loop executions any more because S_1 may be executed an arbitrary number of times. But remember that we assume fairness. This guarantees that S_1 will be executed only finitely many times before S_2 must be activated. In the formalism developed in the previous section the maximal number of times we may neglect S_2 in favour of S_1 is given by $z_2 + 1$ where z_2 is the priority resp. counter variable associated with S_2. The "+1" is necessary here because S_1 may be executed once even if $z_2 = 0$ holds (as we know from the standard invariant IN introduced in Lemma 1). We can summarize this discussion in the following invariant:

$$\begin{aligned} R(\alpha, z_1, z_2) \equiv\ & (\neg A \leftrightarrow \alpha = 0) \\ & \wedge (A \wedge turn = 1 \rightarrow \alpha = 2 + z_2) \\ & \wedge (A \wedge turn = 2 \rightarrow \alpha = 1) \end{aligned}$$

where the underlying well-founded structure is $(\mathbb{N}_0, >)$.

As it turns out $R(\alpha, z_1, z_2)$ indeed satisfies the premises (1)–(4) of the proof rule $FAIR_2$. The soundness of $FAIR_2$ implies the validity of the above correctness formula, i.e. the desired termination result. Note how convenient it is to formalize the informal case analysis with the help of the '*turn*' notation. Also we realize how the additional counter variable z_2 reflects the assumption of fairness. With z_2 we

are able to find an entity α which gets decreased with every loop execution, but without z_2 this is impossible (which accords with the fact that without fairness assumption the program is not guaranteed to terminate).

Let us now look at the simplified rule *S-FAIR*$_2$ where we need an invariant $R^*(\beta, turn)$ with β counting the number of *rounds* only. This time our program analysis is even simpler than the one we had before. If A is false $\beta = 0$ holds. If A is true and we select subprogram S_1 there will be two rounds: $\beta = 2$. Otherwise, if we start with S_2 only one round is possible: $\beta = 1$. This leads to

$$\begin{aligned} R^*(\beta, turn) \equiv\ & (\neg A \leftrightarrow \beta = 0) \\ & \wedge (A \wedge turn = 1 \rightarrow \beta = 2) \\ & \wedge (A \wedge turn = 2 \rightarrow \beta = 1) \end{aligned}$$

where the underlying well-founded structure is simply $(\{0, 1, 2\}, >)$. It is easy to check that $R^*(\beta, turn)$ satisfies the premises of proof rule *S-FAIR*$_2$. Again, the soundness of *S-FAIR*$_2$ implies the desired termination result. Note that it is the fairness assumption which provides a meaningful interpretation of $R^*(\beta, turn)$: only because we know that in the case of "$A \wedge turn = 1$" the first round will be finite we need not bother about *how long* this round will actually be. This kind of reasoning resembles rather closely a *temporal analysis* of the program in the sense of Temporal Logic [13] where one thinks of events (here the end of the first round) which will happen *eventually* without bothering about when exactly.

Example 2.

$$\begin{aligned} \vdash fair \rightarrow \{\mathbf{true}\}\ & \mathbf{do}\ x > 0 \rightarrow A := \mathbf{true} \\ & \square\ x > 0 \rightarrow \mathbf{if}\ A \rightarrow x := x - 1; A := \mathbf{false} \\ & \qquad \square\ \neg A \rightarrow skip \qquad \mathbf{fi} \\ & \mathbf{od}\ \{\mathbf{true}\}. \end{aligned}$$

Encouraged by the previous example we choose to apply the rule *S-FAIR*$_2$ immediately. Thus we have to estimate the number β of rounds again. Observe that now β is not uniformly bounded as in the previous example where $(\beta \leqslant 2)$, but we can give a bound for β depending on the value of the variable x in the initial state: $\beta \leqslant 2x$. Thus we choose $(\mathbb{N}_0, >)$ as a well-founded structure. The following invariant analyzes the possible cases precisely:

$$\begin{aligned} R^*(\beta, turn) \equiv\ & (x \geqslant 0 \leftrightarrow \beta = 0) \\ & \wedge (x > 0 \wedge turn = 1 \rightarrow \beta = 2x) \\ & \wedge (x > 0 \wedge turn = 2 \wedge A \rightarrow \beta = 2x - 1) \\ & \wedge (x > 0 \wedge turn = 2 \wedge \neg A \rightarrow \beta = 2x + 1). \end{aligned}$$

And indeed this invariant suffices to prove the above termination result via proof rule *S-FAIR*$_2$. Recall that every invariant $R^*(\beta, turn)$ for *S-FAIR*$_2$ can be turned into an invariant $R(\alpha, z_1, z_2)$ by the construction of Lemma 2.

Example 3.

$\models fair \to$ {**true**} **do** $x > 0 \to A :=$ **true**;

$\square$ $x > 0 \to$ **if** $B \to x := x + 1$
$\square$ $\neg B \to skip$ **fi**;
$B :=$ **false**;

if $A \to x := x = 1; A :=$ **false**
$\square$ $\neg A \to skip$ **fi**

od {**true**}.

Again we wish to estimate the number of rounds in order to apply rule *S-FAIR*$_2$. But this time we have augmented the program of Example 2 by $\square$ in such a way that even if we know the initial state σ we cannot predict the number of rounds in case that "$x > 0 \wedge B$" holds in σ and the first subprogram is selected. However, as soon as the first round has ended we know the number of the remaining rounds: it is determined exactly as in the previous example. This observation suggests that we now choose as well-founded structure $(\mathbb{N}_0 \cup \{\omega\}, >)$ where $\omega > n$ holds for every $n \in \mathbb{N}_0$, i.e. ω corresponds to the first limit ordinal. Intuitively ω represents the concept of an unknown number which will become precise as soon as ω gets decreased to some $\alpha < \omega$ which must be in $\mathbb{N}_0$. For an initial value of x satisfying $x > 0$ the maximal number β of rounds can now be estimated as

$$(\beta = \omega) \wedge (2x - 1 \leq \beta \leq 2x + 1).$$

With this intuition the following invariant is understandable:

$$\begin{aligned} R^*(\beta, turn) \equiv\ & (x \leq 0 \leftrightarrow \beta = 0) \\ & \wedge (x > 0 \wedge turn = 1 \wedge B \to \beta = \omega) \\ & \wedge (x > 0 \wedge turn = 1 \wedge \neg B \to \beta = 2x) \\ & \wedge (x > 0 \wedge turn = 2 \wedge A \to \beta = 2x - 1) \\ & \wedge (x > 0 \wedge turn = 2 \wedge \neg A \to \beta = 2x + 1). \end{aligned}$$

The part $\square$ is new as compared with the invariant of the previous example. Again $R^*(\beta, turn)$ satisfies the premises of proof rule *S-FAIR*$_2$ and thus proves the desired termination result.

This is the first example which we cannot prove without resorting to infinite ordinals in the well-founded structures even if we count the number of rounds only. (Clearly by counting every single loop execution we would have encountered infinite ordinals already in the previous example. This is so because lexicographically ordered pairs $\alpha = \langle \beta, z \rangle$ needed in the translation of $R^*(\beta, turn)$ into $R(\alpha, z_1, z_2)$ can be written equivalently as

$$\alpha = \omega \cdot \beta + z$$

with $>$ being the usual order between ordinals.) The question arises which ordinals are needed in general to prove total correctness under fairness assumptions. We shall investigate this problem later in Section 11.

5. Fairness: n guards

In this section we extend our approach of dealing with fairness to programs of the form

$$S \equiv \textbf{do } B \to S_1 \,[]\, \cdots \,[]\, B \to S_n \textbf{ od}$$

with n subprograms $S_1, \ldots, S_n$, but still guarded by the same Boolean expression B. Again we proceed in two steps, first looking for a transformation which simulates the fair computations of S and then using this transformation to derive a proof rule for total correctness under fairness assumption. The transformation is a systematic extension of $T^2_{fair}(S)$ from Section 3:

$$\begin{aligned} T^n_{fair}(S) \equiv\ & \textbf{for all } i \in \{1, \ldots, n\} \textbf{ do } z_i := ? \textbf{ od}; \\ & \textbf{do} \\ & []\ B \wedge turn = i \to S_i; z_i := ?; \\ & \qquad \textbf{for all } j \in \{1, \ldots, n\} \backslash \{i\} \textbf{ do } z_j := z_j - 1 \textbf{ od} \\ & \textbf{od} \end{aligned}$$

where i ranges over $\{1, \ldots, n\}$. The $z_1, \ldots, z_n$ are new variables not occurring in S which range over the integers. And the expression "$turn = i$" is an abbreviation defined by

$$turn = i \ \equiv\ i = min\{j \mid z_j = min\{z_k\}_{k=1,\ldots,n}\}$$

which holds if i is the smallest index such that z_i has the minimal value among the $z_1, \ldots, z_n$. As in the case of two guards the variables $z_1, \ldots, z_n$ are *priority variables* used to realize a scheduler in $T^n_{fair}(S)$ which allows only fair computations. At every moment in the computation of $T^n_{fair}(S)$ the subprogram with the smallest index and the maximal priority, say S_i, is executed. After this execution the priorities of all other subprograms S_j, $j \neq i$, get incremented (i.e. z_j decremented) by 1 whereas the priority of S_i gets resets to an arbitrary nonnegative value.

Recall that in the transformed program $T^2_{fair}(S)$ of Section 3 the variables z_1 and z_2 were always $\geqslant -1$. For $T^n_{fair}(S)$ we can show analogously that $z_1, \ldots, z_n \geqslant -n+1$ is always true. In fact, this is a special case of the more general invariant

$$INV \equiv \bigwedge_{i=1}^{n-1} card\{k \mid z_k \leqslant i\} \leqslant n-i$$

which holds in every state of a computation sequence of $T^n_{fair}(S)$. (By $card\, M$ we denote the cardinality of a set M.)

The fact that $T^n_{fair}(S)$ faithfully simulates all the fair computations can be proved in a Lemma analogous to Lemma 1 in Section 3. We shall not do this here because it will be a special case of the more general Lemma 3 in Section 7. But we state

Proposition 2. *Let P and Q be assertions not having $z_1, \ldots, z_n$ as free variables. Then*

$$\models fair \to \{P\}\, S\, \{Q\} \text{ iff } \models \{P\}\, T^n_{fair}(S)\, \{Q\}.$$

which will also follow from Lemma 3.

As in Section 3 this proposition is the starting point for the second step where we apply the proof rule (*) and axiom (**) to $T^n_{fair}(S)$. This leads to the following:

(*FAIR*) *Proof Rule for Fairness: n guards:*

(1) $P \to \forall z_1, \ldots, z_n\, \exists \alpha\, R(\alpha, z_1, \ldots, z_n)$

(2) $\exists \alpha\, R^{INV}(\alpha, z_1, \ldots, z_n) \wedge \neg B \to Q$

(3.i) $i = 1, \ldots, n$:

$\{R^{INV}(\alpha, z_1, \ldots, z_n) \wedge B \wedge turn = i\}$

S_i

$\{\forall z_i\, \exists \beta < \alpha\, R^{INV}(\beta, z_1 - 1, \ldots, z_{i-1} - 1, z_i, z_{i+1} - 1, \ldots, z_n - 1)\}$

(4) $fair \to \{P\}$ **do** $B \to S_1 \,\square \cdots \square\, B \to S_n$ **od** $\{Q\}$

where $z_1, \ldots, z_n$ are new variables not occurring in P, S, Q. The notation $R(\alpha, z_1, \ldots, z_n)$ is used analogously to $R(\alpha, z_1, z_2)$ of Section 3.

$R^{INV}(\alpha, z_1, \ldots, z_n)$ stands for

$$R^{INV}(\alpha, z_1, \ldots, z_n) \equiv R(\alpha, z_1, \ldots, z_n) \wedge INV.$$

Soundness and relative completeness of the rule *FAIR* follows automatically from Proposition 2 and the corresponding results for the proof rule (*) as explained in Section 3. We remark that in the case "$n = 1$", i.e. for $S \equiv$ **do** $B \to S_1$ **od**, the transformed program $T^1_{fair}(S)$ is equivalent to S and the proof rule *FAIR* becomes equivalent to the original proof rule for while-programs.

In the case of two guards we were able to derive a simplified sound rule *S-FAIR*$_2$ from *FAIR*$_2$ which relied only on the notion of a round and was rather easier to apply as shown in Section 4. But in the case of $n \geq 3$ guards this simplification technique does not work successfully any more. Let us explain why. Clearly we could try to generalize the proof rule *S-FAIR*$_2$. This would lead to a rule *S-FAIR* with a premise

(3.*i*) $\{R^*(\beta, i) \wedge B\}$

S_i

$\{(\exists \gamma \leq \beta R^*(\gamma, i)) \wedge \forall j \in \{1, \ldots, n\} \backslash \{i\}\ \exists \gamma < \beta\ R^*(\beta, j)\}$.

The trouble is that even in the simplest cases such an invariant $R^*(\beta, turn)$ satisfying (3.i) may not exist. Here is an example: we would like to prove

$$\models fair \rightarrow \{\textbf{true}\}\ \textbf{do}\ A \rightarrow skip \,\square\, A \rightarrow skip \,\square\, A \rightarrow A := \textbf{false}\ \textbf{od}\ \{\textbf{true}\},$$

a straightforward extension of Example 1 in Section 4. But we cannot do this with the proposed proof rule *S-FAIR* because there is no bound for the number β of rounds if we start with $turn \in \{1, 2\}$ *unless* the priority z_3 of the third subprogram $S_3 \equiv A := \textbf{false}$ is known. Thus we cannot achieve much without priority variables as soon as $n \geq 3$ guards are considered, and therefore must use the original proof rule *FAIR*.

6. Examples for fairness: *n* guards

Let us now examine several applications of the generalized proof rule *FAIR*. We start with the example we could not treat with the proposed simplified rule *S-FAIR*.

Example 4.

$$\models fair \rightarrow \{\textbf{true}\}\ \textbf{do}\ A \rightarrow skip \,\square \cdots \square\, A \rightarrow skip \,\square\, A \rightarrow A := \textbf{false}\ \textbf{od}\ \{\textbf{true}\}.$$

Using the priority variable z_n associated with the nth subprogram $S_n \equiv A := \textbf{false}$ it is quite easy to estimate the maximal number α of loop executions. Due to the standard invariant *INV* we know that for a given value z_n the term $z_n + n - 1$ expresses the maximal number of times the subprogram S_i may be neglected before it will surely be scheduled for execution. Thus if we start with one of the programs $S_1, \ldots, S_{n-1}$ the number α is given by $\alpha = z_n + n$. This is made precise in the following invariant:

$$R(\alpha, z_1, \ldots, z_n) \equiv A \rightarrow \alpha = n + z_n$$

which satisfies the premises of rule *FAIR*. For $n = 2$ this invariant reduces to a simplified version of the invariant $R(\alpha, z_1, z_2)$ we used in Example 1.

Example 5. In the previous example a *particular* subprogram was responsible for terminating the do-od-program, namely the subprogram S_n. We consider now the more general case

$$\begin{aligned}\models \mathit{fair} \to \{\mathbf{true}\}\ \mathbf{do}\ & A_1 \wedge \cdots \wedge A_n \to S_1 \\ \Box\ & A_1 \wedge \cdots \wedge A_n \to S_2 \\ & \cdots\cdots\cdots\cdots \\ \Box\ & A_1 \wedge \cdots \wedge A_n \to S_n \\ & \mathbf{od}\ \{\mathbf{true}\}.\end{aligned}$$

with Boolean variables $A_1, \ldots, A_n$ where the subprograms $S_1, \ldots, S_n$ are partitioned into two subsets $\{S_k \mid k \in K\}$ and $\{S_l \mid l \in L\}$ with $L \neq \emptyset$, $K \cap L = \emptyset$ and $K \cup L = \{1, \ldots, n\}$ such that the following holds: whenever *one* subprogram S_l with $l \in L$ is executed, the whole do-od-program terminates. More specifically, we take

$$S_k \equiv \mathit{skip} \quad \text{for } k \in K \quad \text{and} \quad S_l \equiv A_l := \mathbf{false} \quad \text{for } l \in L.$$

The termination result is proved with the invariant

$$R(\alpha, z_1, \ldots, z_n) \equiv (A_1 \wedge \cdots \wedge A_n) \to \alpha = n + \mathit{min}\{z_l \mid l \in L\}$$

where the minimum of the priority variables z_l is taken because *any* subprogram S_l with $l \in L$ will terminate the do-od-program. A more involved form of this example will reappear later on in Section 10.

Example 6. Finally we study a dual example where *all* subprograms S_i need to be executed in order to terminate the whole do-od-program. The claim is:

$$\begin{aligned}\models \mathrm{fair} \to \{\mathbf{true}\}\ \mathbf{do}\ & A_1 \vee \cdots \vee A_n \to A_1 := \mathbf{false} \quad (S_1) \\ \Box\ & A_1 \vee \cdots \vee A_n \to A_2 := \mathbf{false} \quad (S_2) \\ & \cdots\cdots\cdots\cdots\cdots\cdots \\ \Box\ & A_1 \vee \cdots \vee A_n \to A_n := \mathbf{false} \quad (S_n) \\ & \mathbf{od}\ \{\mathbf{true}\}.\end{aligned}$$

where the Boolean variable A_i reports that the ith subprogram S_i has already been executed.

The prove this claim we have to estimate the maximal number α of runs through the do-od-loop needed to execute every subprogram at least once.

So let us assume that we are currently executing the ith subprogram S_i, i.e. that $\mathit{turn} = i$ holds. Then α depends on how many times we still may neglect the other subprograms S_j, $j \neq i$. This can be measured in terms of the priority variables z_j, $j \neq i$. Taking the standard invariant INV for these variables into account we arrive

at a first estimation of α in case of "$turn = i$":

$$\alpha = n + max\{z_j \mid 1 \leq j \leq n \wedge j \neq i\}.$$

This equation clearly holds at the start of a computation of the do-od-program but is not kept invariant in the course of such a computation. The problem is tha as soon as the jth subprogram S_j has been executed the corresponding priorit variable z_j gets reset arbitrarily whereas the maximum number of times the do-od loop is left to be executed decreases by at least 1. But this deficiency can be fixe easily by using the Boolean expressions A_j which indicate whether S_j has alread been executed. For "$turn = i$" we set

$$R_i \equiv \alpha = n + max\{\textbf{if } A_j \textbf{ then } z_j \textbf{ else } -n+1 \textbf{ fi} \mid 1 \leq j \leq n \wedge j \neq i\}$$

where $-n+1$ is the smallest value z_j can assume. Finally, we define

$$R(\alpha, z_1, \ldots, z_n) \equiv \bigwedge_{i=1}^{n} ((A_1 \vee \cdots \vee A_n) \wedge turn = i \rightarrow R_i).$$

It turns out that $R(\alpha, z_1, \ldots, z_n)$ indeed satisfies the premises of the proof rul *FAIR*.

7. Weak and strong fairness: n guards

We now develop proof rules dealing with fairness for programs of the general forr

$$S \equiv \textbf{do } B_1 \rightarrow S_1 \,\square \cdots \square\, B_n \rightarrow S_n \textbf{ od}$$

with n subprograms guarded by artribrary Boolean expressions. Thus we have t distinguish between weak and strong fairness. Let us start with the *weak fairne* assumption. First we look for a transformation of S which realizes exactly all weak fair computations of S. As in the transformation T^n_{fair} of Section 5 we associate (new) priority variable z_i with every subprogram S_i. But now these variables wi be manipulated individually depending on whether the corresponding guard B_i true or false. Also when determining which of the subprograms $S_1, \ldots, S_n$ is to b executed next we have to make sure that only those priority variables z_i a considered for which the guard B_i is true. These remarks lead to the followir transformation

$T^n_{weak}(S) \equiv$ **for all** $i \in \{1, \ldots, n\}$ **do** $z_i := ?$ **od**;

do

$\square\ B_i \wedge turn = i \rightarrow S_i; z_i := ?;$

for all $j \in \{1, \ldots, n\} \backslash \{i\}$ **do**

(+) $z_j :=$ **if** B_j **then** $z_j - 1$ **else** $?$ **fi**

od

od

where i runs from 1 to n. Here "$turn = i$" is an abbreviation defined by

$$turn = i \equiv i = min\{j \mid z_j = \min\{z_k \mid B_k\}_{k=1,\ldots,n}\}$$

which holds if i is the smallest index for which z_i has the minimal value among all those z_j for which B_j is true. And "**if** B_j **then** $z_j - 1$ **else** ? **fi**" is a conditional expression the value of which is either $z_j - 1$ or an arbitrary nonnegative integer depending on whether B_j is true or false. So we reset the priority variable z_j as soon as B_j is false. This formalizes exactly the weak fairness assumption which tells us that a subprogram S_j is guaranteed to be executed only if the guard B_j is from some moment on continuously true. Note that for $B_1 = \cdots = B_n$ the program $T^n_{weak}(S)$ reduces to $T^n_{fair}(S)$ of Section 5 (except for the last assignment of random values to $z_1, \ldots, z_n$ when the loop is about to terminate).

Note that the scheduler built into S is deterministic in the sense that at each moment only one guard in the transformed do-od-program can be chosen. We could equally well consider nondeterministic schedulers. One possible choice (considered in [12]) is to replace $turn = i$ by a predicate $turn(i) \equiv \forall j(B_j \to z_j \geq z_i)$. All subsequent results then hold, as well.

The following lemma proves the correctness of T^n_{weak} (and hence also of T^n_{fair}) by showing that the transformation is *faithful* in the sense of Section 3.

Lemma 3. (i) *If ξ is a weakly fair computation sequence of S then there exists an extension ξ' of ξ including the new variables $z_1, \ldots, z_n$ such that ξ' is a computation sequence of $T^n_{weak}(S)$.*

(ii) *Conversely, if ξ' is a computation sequence of $T^n_{weak}(S)$ then its restriction ξ to the variables of S is a weakly fair computation sequence of S.*

Proof. (i) Consider a weakly fair computation sequence

$$\xi = \sigma_1 \xrightarrow{i_1} \cdots \sigma_j \xrightarrow{i_j} \cdots$$

of S with $i_j \in \{1, \ldots, n\}$. Analogously to Lemma 1 we show how to extend ξ to a sequence

$$\xi' = \sigma'_1 \xrightarrow{i_1} \cdots \sigma'_j \xrightarrow{i_j} \cdots$$

by assigning new values to the variables $z_1, \ldots, z_n$. This time we define for $l \in \{1, \ldots, n\}$

$$\sigma'_j(z_l) = \begin{cases} min\{k - j \mid k \geq j \wedge i_k = l\} & \text{if } \exists k \geq l : i_k = l, \\ 1 + min\{k - j \mid k \geq j \wedge \models \neg B_l(\sigma_k)\} & \text{otherwise.} \end{cases}$$

We claim that $\sigma'_j(z_l) \in \mathbb{N}_0$ holds for all j and l. To see this we have to show that in both cases the minimum of a non-empty subset of $\mathbb{N}_0$ is taken. By the definition this is true for the case "$\exists k \geq j$: $i_k = l$". So let us assume "$\forall k \geq j$: $i_k \neq l$", i.e. from σ_j on the lth subprogram S_l is never scheduled for execution again. Since ξ is

weakly fair, this can only be the case if $\overset{\infty}{\exists} k \geqslant j : \models \neg B_l(\sigma_k)$ holds. This implies of course $\exists k \geqslant j : \models \neg B_l(\sigma_k)$ which guarantees already that also in this case the minimum is taken over a nonempty set.

Note that in every state σ'_j exactly one variable z_l has the value 0—namely z_{i_j}. So the scheduler built into the program $T^n_{weak}(S)$ will indeed choose the subprogram S_{i_j} in the state σ'_j. Also it is easy to check that the values of the variables $z_1, \ldots, z_n$ in the states σ'_{i_n} are defined in a way which agrees with the corresponding assignments in $T^n_{weak}(S)$. Thus ξ' is indeed a computation sequence of $T^n_{weak}(S)$.

(ii) Conversely, let ξ' be a computation sequence of $T^n_{weak}(S)$ and ξ be its restriction to the variables of S. Clearly ξ is a computation sequence of S, but we have to show that it is weakly fair.

Suppose this is not the case. Then ξ is infinite and one subprogram of S, say S_i, is from a certain state σ_j on, never scheduled for execution though $\models B_i(\sigma_k)$ holds for all $k \geqslant j$. By the definition of $T^n_{weak}(S)$ the value of the variable z_i gets smaller than $-n+1$ in some state σ'_k with $k \geqslant j$. But this is impossible because the assertion

$$INV \equiv \bigwedge_{i=1}^{n} card\{k \mid z_k \leqslant -i\} \leqslant n - i \wedge \bigwedge_{i=1}^{n} [z_i < 0 \to B_i]$$

holds in every state σ'_j of ξ'. We prove this by induction on $j \geqslant 1$. In σ'_1 all $z_1, \ldots, z_n \geqslant 0$ so that INV is vacuously true. Assume now that INV holds in σ'_{j-1}. We show that INV is also true in σ'_j:

(1) Suppose there is some $i \in \{1, \ldots, n\}$ such that there are at least $n-i+1$ indices k for which $z_k \leqslant -i$ holds in σ'_j. Let K be the set of all these indices. By the construction of $T^n_{weak}(S)$, $i \geqslant 2$ and the inequality $z_k \leqslant -i+1$ must have held for all $k \in K$ in the state σ'_{j-1}. By the induction hypothesis $card\, K \leqslant n-i+1$ holds so that indeed $card\, K = n-i+1$. Again by the induction hypothesis $B_k(\sigma'_{j-1})$ is true for all $k \in K$. Hence there is also some $k \in K$ with $i_{j-1} = k$, by the definition of $T^n_{weak}(S)$. But this implies that $z_k \geqslant 0$ holds in σ'_j. This contradicts the definition of K. Consequently $\bigwedge_{i=1}^{n} card\{k \mid z_k \leqslant -i\} \leqslant n-i$ holds in σ'_j.

(2) $\bigwedge_{i=1}^{n} [z_i < 0 \to B_i]$ is obviously true in σ'_j by the construction of $T^n_{weak}(S)$.

This finishes the induction. □

As an immediate consequence of Lemma 3 we derive

Proposition 3. *Let P and Q be assertions without $z_1, \ldots, z_n$ as free variables. The*

$$\models weak \to \{P\}\, S\, \{Q\} \text{ iff } \models \{P\}\, T^n_{weak}(S)\, \{Q\}.$$

Thus an application of the rule (*) and axiom (**) of Section 3 yields the following sound and relatively complete:

(*WEAK*) *Proof Rule for Weak Fairness*: *n guards*

(1) $P \to \forall z_1, \ldots, z_n\, \exists \alpha\, R(\alpha, z_1, \ldots, z_n)$

(2) $\exists \alpha\, R^{INV}(\alpha, z_1, \ldots, z_n) \vee \neg(B_1 \vee \cdots \wedge B_n) \to Q$

$$(3.i)\quad i = 1, \ldots, n:$$

$$\{R^{INV}(\alpha, z_1, \ldots, z_n) \wedge B_i \wedge turn = i\}$$

$$S_i$$

$$\{\forall \tilde{z}_1, \ldots, \tilde{z}_n \forall z_i\ \exists \beta < \alpha$$

$$R^{INV}(\beta, [\textbf{if } B_j \textbf{ then } z_j - 1 \textbf{ else } \tilde{z}_j \textbf{ fi}/z_j]_{j \in \{1,\ldots,n\}\setminus\{i\}})\}$$

$$(4)\quad \frac{}{weak \rightarrow \{P\}\ \textbf{do}\ B_1 \rightarrow S_1 \,\square\, \cdots \,\square\, B_n \rightarrow S_n\ \textbf{od}\ \{Q\}}$$

where $z_1, \ldots, z_n$ are new variables not occurring in P, S, Q. The meaning of "$turn = i$" and $R^{INV}(\alpha, z_1, \ldots, z_n)$ is defined as in Section 5. Note that in the postassertion of premise $(3.i)$ the substitutions are given explicitly in order to shorten the notation. Substituting the conditional expression "**if** B_j **then** $z_j - 1$ **else** ? **fi**" for z_j into R means that in case when B_j holds the assertion R should hold for $z_j - 1$ whereas in case when $\neg B_j$ holds then it should hold for all $z_j \geqslant 0$.

Let us now study the *strong fairness* assumption. The corresponding transformation is:

$$T^n_{strong}(S) \equiv \textbf{for all } i \in \{1, \ldots, n\}\ \textbf{do}\ z_i := ?\ \textbf{od};$$

$$\textbf{do}$$

$$\square\ B_i \wedge turn = i \rightarrow S_i;\ z_i := ?$$

$$\textbf{for all } j \in \{1, \ldots, n\}\setminus\{i\}\ \textbf{do}$$

$$\textbf{if } B_j \textbf{ then } z_j := z_j - 1 \textbf{ fi}$$

$$\textbf{od}$$

$$\textbf{od}$$

where i runs from 1 to n and "$turn = i$" is defined as above. The only difference between $T^n_{weak}(S)$ and $T^n_{strong}(S)$ is that the assignments $z_j := \textbf{if } B_j \textbf{ then } z_j - 1 \textbf{ else } ? \textbf{ fi}$ in line (+) of $T^n_{weak}(S)$ have been replaced by

$$\textbf{if } B_j \textbf{ then } z_j := z_j - 1 \textbf{ fi},$$

i.e. a priority variable z_j of a subprogram S_j which is not executed can never be reset. This realizes the strong fairness assumption by which S_j is guaranteed to be executed when the guard B_j is infinitely often true. Again we can state

Proposition 4. *Let P and Q be assertions without $z_1, \ldots, z_n$ as free variables. Then*

$$\models strong \rightarrow \{P\}\, S\, \{Q\} \text{ iff } \models \{P\}\, T^n_{strong}(S)\, \{Q\}.$$

This proposition relies on a lemma analogous to Lemma 3. There are two things we have to alter in the proof of Lemma 3 so that it works for strong fairness instead

of weak fairness. First is the definition of the new values $\sigma'_j(z_l)$ of the variables z_l, $l=1,\ldots,n$ in the extended computation sequence ξ' of ξ considered in part (i). This definition now reads

$$\sigma'_j(z_l)=\begin{cases} card\{m \mid j<m\leqslant k_0 \wedge \models B_l(\sigma_m)\} & \text{if } \exists k\geqslant j: i_k=l,\\ 1+card\{m \mid j\leqslant m \wedge \models B_l(\sigma_m)\} & \text{otherwise}\end{cases}$$

where $k_0=min\{k\geqslant j \wedge i_k=l\}$. Again it is not difficult to see that these values are well-defined if ξ is strongly fair and consistent with the transformation T^n_{strong}.

Secondly, in part (ii) of the proof of Lemma 3 we have rather to consider the following assertion

$$INV \equiv \bigwedge_{i=1}^{n} card\{k \mid z_k\leqslant -i\}\leqslant n-i$$

$$\wedge \bigwedge_{i=1}^{n-1} (card\{k \mid z_k\leqslant -1\}=n-i \rightarrow \bigvee_{k=1}^{n} (z_k\leqslant -i \wedge B_k)).$$

The proof of (1) remains virtually unchanged. The proof of (2) runs now as follows. We refer here to the notation and proof of (1) given in the proof of Lemma 3.

Consider some $i\in\{1,\ldots,n-1\}$ such that $card\,K=n-i$ holds for $K=\{k \mid z_k\leqslant -i\}$ in σ'_j. We have to show then that B_k is true in σ'_j for some $k\in K$. As in (1) we conclude that $z_k\leqslant -i+1$ holds for all $k\in K$ in σ'_{j-1}. It is impossible that $z_k\leqslant -i$ holds already for all $k\in K$ in σ'_{j-1}. Because then by the induction hypothesis B_k would be true for some $k\in K$ in σ'_{j-1}. Thus by the construction of $T^n_{strong}(S)$ there would also be some $k\in K$ with $i_{j-1}=k$ and $z_k\geqslant 0$ in σ'_j which contradicts the definition of K. So there exists some $k\in K$ with $z_k=-i+1$ in σ'_{j-1}. But then the value of z_k decreased so B_k is indeed true for this k in σ'_j by definition of $T^n_{strong}(S)$.

Proposition 4 gives rise to the following sound and relatively complete:

(STRONG) Proof Rule for Strong Fairness: n guards:

(1) $P\rightarrow \forall z_1,\ldots,z_n\ \exists\alpha\, R(\alpha,z_1,\ldots,z_n)$

(2) $\exists\alpha\, R^{INV}(\alpha,z_1,\ldots,z_n)\wedge\neg(B_1\vee\cdots\vee B_n)\rightarrow Q$

(3.*i*) $i=1,\ldots,n$:

$\{R^{INV}(\alpha,z_1,\ldots,z_n)\wedge B_i\wedge turn=i\}$

S_i

$\{\forall z_i\ \exists\beta<\alpha\, R^{INV}(\beta,[\textbf{if } B_j \textbf{ then } z_j-1 \textbf{ else } z_j \textbf{ fi}]_{j\in\{1,\ldots,n\}\setminus\{i\}})\}$

(4) $strong\rightarrow\{P\}\ \textbf{do}\ B_1\rightarrow S_1\,\square\cdots\square\, B_n\rightarrow S_n\ \textbf{od}\ \{Q\}$

with the same conventions as for proof rule *WEAK*.

Considering the framework of do-od-programs S with different two guards B_1, B_2 we can try to find simplified proof rules relying on the intuition of a *round* as we did earlier in Section 3 for the case of two identical guards. We show such a simplified version for the case of strong fairness.

(*S-STRONG*$_2$) *Simplified Proof Rule for Strong Fairness*: 2 *guards*:

(1) $P \to \forall\, turn\ \exists \beta\, R^*(\beta, turn)$

(2) $\exists \beta\, R^*(\beta, turn) \wedge \neg(B_1 \vee B_2) \to Q$

(3) $\{R^*(\beta, 1) \wedge B_1\}$

S_1; **do** $B_1 \wedge \neg B_2 \to S_1$ **od**

$\{\exists \gamma \leqslant \beta\, R^*(\gamma, 1) \wedge \exists \gamma < \beta\, R^*(\gamma, 2)\}$

(4) $\{R^*(\beta, 2) \wedge B_2\}$

S_2; **do** $B_2 \wedge \neg B_1 \to S_2$ **od**

$\{\exists \gamma \leqslant \beta\, R^*(\gamma, 2) \wedge \exists \gamma < \beta\, R^*(\gamma, 1)\}$

---

strong $\to \{P\}$ **do** $B_1 \to S_1 \,\square\, B_2 \to S_2$ **od** $\{Q\}$

where *turn* is a new variable not occurring in P, B_1, S_1, B_2, S_2, Q which ranges over the set $\{1, 2\}$.

Note that in this rule we have added loops **do** $B_i \wedge \neg B_{3-i} \to S_i$ **od** behind S_i $(i = 1, 2)$. These loops are intended to absorb all the cases where we know in advance what the strongly fair scheduler would recommend us to do, namely to continue the execution of S_i. We remark that the addition of these loops is not necessary but it leads to a proof rule which is easier to apply as we shall see in the next section.

The soundness of *S-STRONG*$_2$ can be proved analogously to that of *S-FAIR*$_2$, i.e. by factorizing the original α used in *STRONG* into a pair $\alpha = \langle \beta, z \rangle$ where β gets decreased only if there is no possibility to decrease z. That the factor β can indeed be interpreted as counting the number of rounds depends critically on the additional loop **do** $B_1 \wedge \neg B_2 \to S_1$ **od**. Without this loop we would be forced to decrease β in the postassertion of premise (3) even when $B_1 \wedge \neg B_2$ holds and therefore no new round begins. And analogously for **do** $B_2 \wedge \neg B_1 \to S_2$ **od**.

8. Examples for strong fairness

In this section we apply the simplified proof rule *S-STRONG*$_2$ for strong fairness to prove the correctness of programs which terminate under strong but not under weak fairness assumptions. These programs are closely related to those studied in Section 4.

Example 7.

$$\models \mathit{strong} \to \{\mathbf{true}\}\ \mathbf{do}\ A \to B := \neg B$$
$$\square\ B \to A := \mathbf{false};\ B := \mathbf{false}$$
$$\mathbf{od}\ \{\mathbf{true}\}.$$

This is a refined version of Example 1. Again the number β of rounds is uniformly bounded and independent of the initial state. We choose as well-founded structure $(\{0, 1, 2\}, >)$ and as invariant

$$R^*(\beta, \mathit{turn}) \equiv (\neg A \wedge \neg B \leftrightarrow \beta = 0)$$
$$\wedge\ (A \wedge \mathit{turn} = 1 \to \beta = 2)$$
$$\wedge\ (B \wedge \mathit{turn} = 2 \to \beta = 1).$$

It turns out that $R^*(\beta, \mathit{turn})$ satisfies the premises of the rule *S-STRONG*$_2$. Note that the do-od-loops to be considered in the premises (3) and (4) are simply

$$\mathbf{do}\ A \wedge \neg B \to B := \neg B\ \mathbf{od}$$

and

$$\mathbf{do}\ B \wedge \neg A \to A := \mathbf{false};\ B := \mathbf{false}\ \mathbf{od}$$

which terminate after at most one iteration.

What happens now if we assume only *weak* fairness? Then an infinite computation sequence

$$\xi = \sigma_1 \xrightarrow{1} \sigma_2 \xrightarrow{1} \cdots \sigma_j \xrightarrow{1} \cdots$$

is still possible because there is no state σ_j such that the guard B is continuously true from σ_j on and hence weak fairness does not force us to eventually choose the second subprogram. Consequently the program may diverge under weak fairness assumption.

Example 8.

$$\models \mathit{strong} \to \{\mathbf{true}\}\ \mathbf{do}\ x > 0 \wedge \boxed{C} \to A := \mathbf{true};\ \boxed{D := \neg D}$$
$$\square\ x > 0 \wedge \boxed{D} \to \mathbf{if}\ A \to x := x - 1;\ A := \mathbf{false}$$
$$\square \neg A \to \mathit{skip}\quad \mathbf{fi};$$
$$\boxed{C := \neg C}$$
$$\mathbf{od}\ \{\mathbf{true}\}.$$

This is a refined version of Example 2, only the $\boxed{}$ parts have been added.

Thus our observations about the number of rounds remain valid. This can also be seen from the corresponding invariant

$$\begin{aligned} R^*(\beta, turn) \equiv \quad & (x \leqslant 0 \mid \vee (\neg C \wedge \neg D) \leftrightarrow \beta = 0) \\ & \wedge (x > 0 \mid \wedge C \mid \wedge turn = 1 \rightarrow \beta = 2x) \\ & \wedge (x > 0 \mid \wedge D \mid \wedge turn = 2 \wedge A \rightarrow \beta = 2x - 1) \\ & \wedge (x > 0 \mid \wedge D \mid \wedge turn = 2 \wedge \neg A \rightarrow \beta = 2x + 1) \end{aligned}$$

where the underlying well-founded structure is $(\mathbb{N}_0, >)$. It differs from the invariant of Example 2 only by the $\boxed{\quad}$ part. Again the termination result does not hold under weak fairness assumption.

Example 9.

$$\begin{aligned} \models strong \rightarrow \{\mathbf{true}\}\ \mathbf{do}\ x > 0 \mid \wedge C \mid & \rightarrow A := \mathbf{true}; \\ & \quad \mathbf{if}\ B \rightarrow x := x + 1 \\ & \quad \square\ \neg B \rightarrow skip\ \mathbf{fi}; \\ & \quad D := \neg D \\ \square\ x > 0 \mid \wedge D \mid & \rightarrow B := \mathbf{false}; \\ & \quad \mathbf{if}\ A \rightarrow x := x - 1; A := \mathbf{false} \\ & \quad \square\ \neg A \rightarrow skip \qquad \mathbf{fi}; \\ & \quad C := \neg C \\ \mathbf{od}\ \{\mathbf{true}\}. & \end{aligned}$$

Also this termination result does not hold under weak fairness assumption. Here the same refinement technique as in the previous example has been applied to Example 3. This leads us to the following invariant:

$$\begin{aligned} R^*(\beta, turn) \equiv \quad & (x \leqslant 0 \mid \vee (\neg C \wedge \neg D) \leftrightarrow \beta = 0 \\ & \wedge (x > 0 \mid \wedge C \mid \wedge turn = 1 \wedge B \rightarrow \beta = \omega) \\ & \wedge (x > 0 \mid \wedge C \mid \wedge turn = 1 \wedge \neg B \rightarrow \beta = 2x) \\ & \wedge (x > 0 \mid \wedge D \mid \wedge turn = 2 \wedge A \rightarrow \beta = 2x - 1) \\ & \wedge (x > 0 \mid \wedge D \mid \wedge turn = 2 \wedge \neg A \rightarrow \beta = 2x + 1) \end{aligned}$$

where we use the well-founded structure $(\mathbb{N}_0 \cup \{\omega\}, >)$ known from Example 3.

Note that in the above examples we may switch turns only after every even number of computation steps. One might think that more elaborated 'switching'

techniques lead to more complicated invariants $R^*(\beta, turn)$. Fortunately this is not the case. For example, if we replace in the above program the conjunct "$\wedge C$" in the first guard by "$\wedge y = 0$" and the assignment "$C := \neg C$" by "$y := (y + 1 \textbf{ mod } 5)$", the only thing we have to alter in $R^*(\beta, turn)$ is to change "C" into "$y = 0$". Our proof rule *S-STRONG*$_2$ is robust against such changes in the switching technique thanks to the extra do-od-loops in their premises (3) and (4). These loops absorb the complexity of the new switching technique.

9. Zero searching

So far our examples were somewhat artificial and served only as means to identify certain classes of fair computation sequences. Also all the examples dealt with the issue of termination only. We now prove total correctness of a program which is interesting for its own sake. The claim is that

$$\models fair \rightarrow \{x = y = w \wedge \exists u\, B(u)\}\, S\, \{B(w)\}$$

holds where

$$\begin{aligned} S \equiv\ & \textbf{do } \neg B(x) \wedge \neg B(y) \rightarrow x := x + 1;\ w := x \\ & \square\ \neg B(x) \wedge \neg B(y) \rightarrow y := y - 1;\ w := y \\ & \textbf{od}. \end{aligned}$$

Here $\models$ means validity in the standard interpretation of integers. B is a Boolean expression with a free variable u, but without occurrences of the variables x, y, w. A useful choice would be

$$B(u) \equiv f(u) = 0$$

where f is a function from integers into integers. Then the program S searches for a zero of f. It does so by employing two subprograms, one is searching for this zero by continuously incrementing its test values ($x := x + 1$) and the other one by decrementing them ($y := y - 1$). The idea is that S finds the desired zero by activating these subprograms in a nondeterministic, but fair order. The formal correctness proof of S will be divided into three steps.

Step 1. We first show that S works correctly under the assumption that initially "$B(u) \wedge x \leq u$" holds for some 'zero' u:

$$\models fair \rightarrow \{x = y = w \wedge B(u) \wedge x \leq u\}\, S\, \{B(w)\}.$$

We intend to apply the simplified proof rule *S-FAIR*$_2$ of Section 3. Note that the maximal number β of rounds of S depends on the difference $u - x$ between u and x. It is $\beta = 2(u - x)$ if $turn = 2$ holds and $\beta = 2(u = x) - 1$ if $turn = 1$ holds. Thus we

take $(\mathbb{N}_0, >)$ as well-founded structure. The precise formulation of the invariant is

$$\begin{aligned} R_1^*(\beta, turn) \equiv\ & B(u) \wedge x \leqslant u \\ & \wedge (B(x) \vee B(y) \rightarrow B(w)) \\ & \wedge (\neg B(x) \wedge \neg B(y) \wedge turn = 1 \rightarrow \beta = 2(u-x)-1) \\ & \wedge (\neg B(x) \wedge \neg B(y) \wedge turn = 2 \rightarrow \beta = 2(u-x)). \end{aligned}$$

We show that $R_1^*(\beta, turn)$ satisfies the premises (1)–(4) of the proof rule *S-FAIR*$_2$. It is easy to see that $R_1^*(\beta, turn)$ satisfies the premises (1) and (2) with $P \equiv (x = y = w \wedge B(u) \wedge x \leqslant u)$ and $Q \equiv B(w)$. Slightly more care is needed to verify premise (3). Define P_1 and Q_1 as follows:

$$\begin{aligned} P_1 &\equiv (B(u) \wedge \neg B(x) \wedge \neg B(y) \wedge x < u \wedge \beta \geqslant 2(u-x)-1 \geqslant 1) \\ Q_1 &\equiv B(u) \wedge x \leqslant u \wedge \beta > 2(u-x) \\ &\quad \wedge (B(x) \vee B(y) \rightarrow B(w)). \end{aligned}$$

Then $\models \{P_1\}\, x := x+1;\ w := x\ \{Q_1\}$ holds. An application of the *rule of consequence* yields the premise (3). A similar argument verifies premise (4).

Step 2. Next we replace the assumption "$B(u) \wedge x \leqslant u$" by "$B(u) \wedge u \leqslant y$":

$$\models fair \rightarrow \{x = y = w \wedge B(u) \wedge u \leqslant y\}\, S\, \{B(w)\}.$$

Since this claim is symmetric to that of Step 1, we can easily derive the corresponding invariant $R_2^*(\beta, turn)$ from $R_1^*(\beta, turn)$:

$$\begin{aligned} R_2^*(\beta, turn) \equiv\ & B(u) \wedge u \leqslant y \\ & \wedge (B(x) \vee B(y) \rightarrow B(w)) \\ & \wedge (\neg B(x) \wedge \neg B(y) \wedge turn = 1 \rightarrow \beta = 2(y-u)) \\ & \wedge (\neg B(x) \wedge \neg B(y) \wedge turn = 2 \rightarrow \beta = 2(y-u)-1). \end{aligned}$$

Step 3. We combine the results of Steps 1 and 2 into

$$\models fair \rightarrow \{x = y = w \wedge B(u) \wedge (x \leqslant u \vee u \leqslant y)\}\, S\, \{B(w)\}.$$

Notice that the preassertion can be replaced by the equivalent assertion "$x = y = w \wedge B(u)$". Finally, we can prefix $B(u)$ in this assertion by the existential quantifier $\exists u$ because u occurs neither in S nor in the post-assertion $B(w)$. This yields the desired result:

$$\models fair \rightarrow \{x = y = w \wedge \exists u B(u)\}\, S\, \{B(w)\}.$$

These last steps correspond to applications of some general rules for reasoning about correctness formulas, namely the *disjunction rule*, the *rule of consequence*, and the *∃-rule*. These rules are sound regardless of whether fairness is assumed or not.

10. Asynchronous fixed point computation

In this section we study a natural example for fairness with $n \geq 2$ guards suggested by P. Cousot. Let $(\mathscr{L}, \subseteq)$ be a complete lattice which fulfils the *finite chain property*, i.e. every strictly increasing chain

$$x_1 \subset x_2 \subset x_3 \subset \cdots$$

in $\mathscr{L}$ is finite. Thus $(\mathscr{L}, \subset)$ is a well-founded structure. Then the product $(\mathscr{L}^n, \subseteq)$ with $n \geq 2$ is also a complete lattice with the finite chain property. We consider now a monotonic w.r.t. $\subseteq$ operator

$$F : \mathscr{L}^n \to \mathscr{L}^n.$$

By Knaster–Tarski's theorem F has a least fixed point $\mu F \in \mathscr{L}^n$. We wish to compute μF asynchronously by employing n subprograms S_i each of which is allowed only to apply the ith component function

$$F_i : \mathscr{L}^n \to \mathscr{L}$$

of F defined by $F_i(x_1, \ldots, x_n) = y_i$ where $F(x_1, \ldots, x_n) = (y_1, \ldots, y_n)$. These subprograms are activated nondeterministically by the following program:

$$S \equiv \textbf{do } B \to x_1 := F_1(\bar{x}) \,\square\, \cdots \,\square\, B \to x_n := F_n(\bar{x}) \textbf{ od}$$

where $\bar{x} \equiv (x_1, \ldots, x_n)$ and $B \equiv \neg(\bar{x} = F(\bar{x}))$. In general S of course will not compute μF but the claim is that it will do so under the assumption of fairness:

(C) $\quad \models \mathit{fair} \to \{\bar{x} = \bot\}\, S\, \{\bar{x} = \mu F\}$

where $\bot$ is the least element in $\mathscr{L}^n$ and $\models$ refers to the validity in $\mathscr{L}$. (This correctness result is a special case of a more general theorem proved in Cousot [4].) We would like to prove (C) with the help of the proof rule *FAIR*. To this end we proceed in two steps.

Step 1. We start with an informal analysis of program S. Consider a computation sequence

$$\xi = \sigma_1 \xrightarrow{i_1} \cdots \sigma_j \xrightarrow{i_j} \cdots$$

of S and define $\sigma_j(\bar{x}) \equiv (\sigma_j(x_1), \ldots, \sigma_j(x_n))$ for $j \geq 1$ and

$$F_i[\bar{x}] = (x_1, \ldots, x_{i-1}, F_i(\bar{x}), x_{i+1}, \ldots, x_n)$$

for $1 \leq i \leq n$. Since $\sigma_1(\bar{x}) = \bot$ holds and the component functions F_i are monotonic the assertion

(*) $\quad \bot \subseteq \bar{x} \subseteq F_i[\bar{x}] \subseteq \mu F$

is true for every $i \in \{1, \ldots, n\}$ in every state σ_j of ξ. Thus $\bar{x} = \mu F$ will hold as soon as S has terminated with $\bar{x} = F(\bar{x})$. But why does S terminate? Note that by (*

the program S produces an increasing chain

$$\sigma_1(\bar{x}) \subseteq \cdots \subseteq \sigma_j(\bar{x}) \subseteq \cdots$$

of values in the variables $\bar{x}$. That there exists some state σ_j with $\bar{x} = F(\bar{x})$ relies on two facts:

(1) By the finite chain property of $\mathscr{L}$ resp. $\mathscr{L}^n$ the values $\sigma_j(\bar{x}) \in \mathscr{L}^n$ cannot be increased infinitely often.

(2) And by the fairness assumption the values $\sigma_j(\bar{x})$ cannot be constant arbitrarily long without increasing.

(1) is clear, but (2) needs a proof. Consider some non-terminal state σ_j in ξ (i.e. satisfying $B \equiv \neg(\bar{x} = F(\bar{x}))$) for which either $\sigma_j = \sigma_1$ (start) or $\sigma_{j-1}(\bar{x}) \subset \sigma_j(\bar{x})$ (increase just happened) holds. Then we can find two index sets K and L—both depending on σ_j—which partition the subprograms $S_1, \ldots, S_n$ of S into subsets $\{S_k \mid k \in K\}$ and $\{S_l \mid l \in L\}$ such that the S_k stabilize the values of $\bar{x}$, i.e. $\bar{x} = F_k[\bar{x}]$ holds for $k \in K$ in σ_j, whereas the S_l increase the values of $\bar{x}$, i.e. $\bar{v} \subset F_l[\bar{x}]$ holds for $l \in L$ in σ_j. (Note that $L \neq \emptyset$ holds because σ_j is non-terminal.)

Thus as long as subprograms S_k with $k \in K$ are executed, the do-od-program S generates states $\sigma_{j+1}, \sigma_{j+2}, \ldots$, satisfying

$$\sigma_j(\bar{x}) = \sigma_{j+1}(\bar{x}) = \sigma_{j+2}(\bar{x}) = \cdots.$$

But as soon as a subprogram S_l with $l \in L$ is executed in some state σ_m with $j \leq m$, we get the desired next increase

$$\sigma_m(\bar{x}) \subset \sigma_{m+1}(\bar{x})$$

after σ_j. That such an increase will indeed happen depends on the assumption of fairness. The formal proof of this fact is rather close to that of Example 5, except for the following changes:

- Instead of proving termination like in Example 5 we are now proving the increase of the values of $\bar{x}$.
- This increase will be accomplished by executing one subprogram $S_l \equiv x_l := F_l(x_l)$ with $l \in L$ instead of just setting a Boolean variable A_l to **false** as in Example 5.
- The index sets K and L vary now with the states σ_j, i.e. to verify program S we rely on different instances of the argument given in Example 5.

Step 2. With this informal discussion in mind we are now prepared for the formal correctness proof of S with proof rule *FAIR*. As well-founded structure we choose $(\mathscr{L}^n \times \mathbb{N}_0, >)$ where $>$ is the lexicographic order defined by

$$\langle \bar{x}_1, n_1 \rangle > \langle \bar{x}_2, n_2 \rangle \quad \text{if } \bar{x}_1 \subset \bar{x}_2 \vee (\bar{x}_1 = \bar{x}_2 \wedge n_1 > n_2).$$

Since $\mathscr{L}^n$ has the finite chain property, $>$ is clearly well-founded. The components $\bar{x}$ and n of pairs $\langle \bar{x}, n \rangle \in \mathscr{L}^n \times \mathbb{N}_0$ correspond to the facts (1) resp. (2) about

termination of S explained above. The right invariant is now

$$R(\alpha, z_1, \ldots, z_n) \equiv \bigwedge_{i=1}^{n} (\bot \subseteq \bar{x} \subseteq F_i[\bar{x}] \subseteq \mu F)$$
$$\wedge (B \to \alpha = \langle \bar{x}, n + min\{z_l \mid 1 \leq l \leq n \wedge \bar{x} \subset F_l[\bar{x}]\}\rangle)$$

where the expression in the second component of α is a suitably adjusted version of that from Example 5. Note that the Boolean guard B being **true** guarantees that there exists some $l \in \{1, \ldots, n\}$ with $\bar{x} \subset F_l[\bar{x}]$. So we need not bother how to define the minimum over empty sets here.

Clearly $R(\alpha, z_1, \ldots, z_n)$ satisfies the premises (1) and (2) of the proof rule *FAIR*. To check the premise (3.i) consider the execution of S_i under the precondition

$$R^{INV}(\alpha, z_1, \ldots, z_n) \wedge B \wedge turn = i,$$

in particular with

$$\alpha = \langle \bar{x}, n + min\{z_l \mid 1 \leq l \leq n \wedge \bar{x} \subset F_l[\bar{x}]\}\rangle.$$

Let us denote by $\bar{x}'$ the values of $\bar{x}$ after the execution of S_i. To establish the postcondition, we have to show that for every $z_i \geq 0$

$$\exists \beta < \alpha \, R^{INV}(\beta, z_1 - 1, \ldots, z_{i-1} - 1, z_i, z_{i+1} - 1, \ldots, z_n - 1)$$

holds. There are two cases to be considered.

(i) $\bar{x} \subset \bar{x}' = F_i[\bar{x}]$: Then α gets decreased to β by its first component.

(ii) $\bar{x} = \bar{x}' = F_i[\bar{x}]$: Since B held before the execution of S_i, there exist indices $l \in \{1, \ldots, n\}$ with $\bar{x} \subset F_l[\bar{x}]$. In fact, $l \neq i$ holds for all such indices due to $\bar{x} = F_i[\bar{x}]$.

Thus the following definition of β is independent of z_i:

$$\beta = \langle \bar{x}, n + min\{z_l - 1 \mid 1 \leq l \leq n \wedge \bar{x} \subset F_l[\bar{x}]\}\rangle.$$

Clearly $\beta < \alpha$, and due to $\bar{x} = \bar{x}'$, the invariant $R^{INV}(\beta, z_1 - 1, \ldots, z_{i-1} - 1, z_i, z_{i+1} - 1, \ldots, z_n - 1)$ holds after the execution of S_i for every $z_i \geq 0$.

Thus in both cases α gets decreased.

Finally, we observe that for this example it was very convenient to have arbitrary well-founded structures at our disposal. If we were restricted to their standard representation as ordinals we would have to run into difficulties when dealing with the not further analyzed structure of $\mathscr{L}^n$.

11. All ordinals $\alpha < \omega \cdot \omega^\omega$ are necessary

In this section we investigate the question raised in Section 4, namely which ordinals α are necessary for proving total correctness of programs

$$S \equiv \textbf{do } B_1 \to S_1 \,\square \cdots \square\, B_n \to S_n \textbf{ od}$$

under fairness assumptions. More precisely we ask what is the smallest ordinal α such that the well-founded structure $(W, >_w)$ needed in the correctness proof of S can be embedded into $(W_\alpha, >)$. (Cf. Section 3 for the notation.) We show that *at least* all ordinals

$$\alpha < \omega.\omega^\omega$$

are needed. To this end we explicitly construct for every $n \in \mathbb{N}$ a program S_{ω^n} of the form

$$S_{\omega^n} \equiv \textbf{do } B \to S_1 \,\square\, \cdots \,\square\, B \to S_n \textbf{ od}$$

such that the ordinal $\beta = \omega^n$ is needed to count the number of *rounds* which may occur in fair computations of S_{ω^n}. Thus at least all ordinals $\beta < \omega^\omega$ are needed to count the number of rounds in programs S. This implies the claim about α by Lemma 2. (We remark that our result has been improved considerably in [3] with help of general recursion theoretic methods. In [3] it is shown that *exactly all recursive ordinals* α are necessary to deal with programs S under fairness assumptions. Nevertheless it seems worthwhile to explain the simple structure of the programs S_{ω^n}.)

The idea behind the construction of these S_{ω^n} is simple. It can be illustrated by certain programs T_{ω^n} which use random assignments—which are of course disallowed within the components S_i of S—and n nested loops. These programs T_{ω^n} are constructed in such a way that the ordinal associated with the total number of executions of all nested loops is ω^n. We define

$$\begin{aligned}
T_\omega \equiv\ & x_1 := ?; \\
& \textbf{do } x_1 > 0 \to x_1 := x_1 - 1 \textbf{ od} \\
T_{\omega^2} \equiv\ & x_1 := ?; \\
& \textbf{do } x_1 > 0 \to x_2 := ?; \\
& \qquad \textbf{do } x_2 > 0 \to x_2 := x_2 - 1 \textbf{ od}; \\
& \qquad x_1 := x_1 - 1 \\
& \textbf{od}
\end{aligned}$$

Once x_1 is fixed, the outer loop is executed x_1 times. Within each such execution we arbitrarily choose x_2 and execute the inner loop x_2 times. Thus the ordinal representing the total number of executions of both loops is ω^2. It is obvious how to obtain $T_{\omega^3}, \ldots, T_{\omega^n}$ for $n \in \mathbb{N}$.

Now we translate this idea into the framework of fair, nondeterministic programs S such that ω^n becomes the ordinal representing the number of rounds instead of the number of loop executions. We do this by inductively defining programs S_{ω^n}, $n \in \mathbb{N}$:

For $n = 1$ we have

$$S_\omega = \textbf{do } x_1 > 0 \wedge 1 \leq q \leq 3 \rightarrow \textbf{if } q = 1 \rightarrow x_1 := x_1 + 1$$
$$\square\ q = 2 \rightarrow skip$$
$$\boxed{\square\ q = 3 \rightarrow q := 2}$$
$$\textbf{fi}$$
$$\square x_1 > 0 \wedge 1 \leq q \leq 3 \rightarrow \textbf{if } q = 1 \rightarrow q := 3$$
$$\square\ q = 2 \rightarrow x_1 := x_1 - 1; q := 3$$
$$\boxed{\square\ q = 3 \rightarrow skip}$$
$$\textbf{fi}$$
$$\textbf{od}$$

where x_1 and q are integer variables.

The effect of the random assignments $x_1 := ?$ is achieved here by repeate execution of the first component in case when $q = 1$. Switching to the secon component results in a change of q to 3. From that moment on x_1 is gradual decreased by one—once for two rounds.

Obviously, S_ω is closely related to the program of Example 3. Thus it is easy t understand that $\beta = \omega$ is needed to count the number of rounds.

Let $n > 1$ and $S_{\omega^{n-1}}$ be known. In $S_{\omega^{n-1}}$ the integer variables $x_1, \ldots, x_n$ and occur. The guards of both subcomponents in $S_{\omega^{n-1}}$ are "$x_1 > 0 \wedge \cdots \wedge x_{n-1} \geq 0 \wedge 1$ $q \leq 2(n-1)+1$". Accordingly, both subcomponents are if-fi-clauses of the form

$$\textbf{if } q = 1 \rightarrow \cdots \square \cdots \square q = 2(n-1)+1 \rightarrow \cdots \textbf{ fi}.$$

With this structure of $S_{\omega^{n-1}}$ in mind, the following definition of S_{ω^n} makes sens S_{ω^n} is obtained from $S_{\omega^{n-1}}$ by replacing

(1) both guards

$$"x_1 > 0 \wedge x_2 \geq 0 \wedge \cdots \wedge x_{n-1} \geq 0 \wedge 1 \leq q \leq 2(n-1)+1"$$

by

$$"x_1 > 0 \wedge x_2 \geq 0 \wedge \cdots \wedge x_n \geq 0 \wedge 1 \leq q \leq 2n+1";$$

(2) the part $\boxed{\square q = 2(n-1)+1 \rightarrow q := 2(n-1)}$ by

$$\square\ q = 2(n-1)+1 \rightarrow q := 2n+1$$

$$\square\ q = 2n \wedge x_n > 0 \rightarrow x_n := x_n - 1; q := 2n+1$$

$$\square\ q = 2n \wedge x_n = 0 \rightarrow q := 2(n-1)$$

$$\square\ q = 2n+1 \rightarrow skip$$

(3) the part $\square q = 2(n-1)+1 \to skip$ by

$$\square\ q = 2(n-1)+1 \to x_n := x_n + 1$$

$$\square\ q = 2n \to skip$$

$$\square\ q = 2n+1 \to q := 2n$$

where x_n is a new integer variable.

As an example let us consider S_{ω^2}.

S_{ω^2} ≡ **do** $x_1 > 0 \wedge x_2 \geqslant 0 \wedge 1 \leqslant q \leqslant 5 \to$

if $q = 1 \to x_1 := x_1 + 1$

$\square\ q = 2 \to skip$

$\square\ q = 3 \to q := 5$

$\square\ q = 4 \wedge x_2 > 0 \to x_2 := x_2 - 1;\ q := 5$

$\square\ q = 4 \wedge x_2 = 0 \to q := 2$

$\square\ q = 5 \to skip$

fi

$\square\ x_1 > 0 \wedge x_2 \geqslant 0 \wedge 1 \leqslant q \leqslant 5 \to$

if $q = 1 \to q := 3$

$\square\ q = 2 \to x_1 := x_1 - 1;\ q := 3$

$\square\ q = 3 \to x_2 := x_2 + 1$

$\square\ q = 4 \to skip$

$\square\ q = 5 \to q := 4$

fi

od

To see that ordinal $\beta = \omega^2$ is indeed to represent the number of rounds, let us choose component 1 of S_{ω^2} in a state where $q = 1$ holds. Thus in the first round the value of x_1 is determined. Subsequently x_1 is gradually decreased by 1 in component 2 at $q = 2$. Each time x_1 is decreased, the variable x_2 is set to some arbitrary value at $q = 3$. This value x_2 is then gradually decreased by 1 in component 1 at $q = 4$ and after each decrease control is eventually switched to component 2 (and then back again). Summarizing, arbitrarily often (x_1 times) we have a switching

phase with arbitrarily many switches (x_2 times each phase). This yields indeed $\beta = \omega^2$ as resulting ordinal.

12. Conclusions

We presented here a simple proof theoretic approach to fairness. This approach is operational in nature because an explicit program transformation is at the source of each rule. However, this transformation is reflected in the assertions and not in the program considered. Thus the original program is studied and not its translated version.

Transformations realizing fairness first appeared in [1]. These transformations were unnecessarily complex. Nevertheless the idea inspired other work like [12], [6] and [3]. Present simplifications reflect our better understanding of the rôle of the transformations.

We hope that this approach sheds some new light on the use of program transformations in the context of proof systems for program correctness. Such transformations are usually (and rightly!) criticized because of the resulting syntactic explosion of the programs considered (see for example the discussion following [7]). But in our case however, no syntactic explosion arises as a different type of transformation is used: whereas in [7] and [8] the structure of the program is destroyed by dividing the components of parallel programs into single individual actions, our transformations add only parts for scheduling purposes.

In principle our approach can be extended to deal with programs allowing nested nondeterminism where fairness is required at all levels—not only the top one. Then our transformations would simply be applied in a nested fashion, but provided all initializations $z := ?$ would be moved to the beginning of the transformed program. The reason that we cannot do here without these initializations is because in our approach the priority variables z can become negative, and we need the invariant *INV* to ensure that they cannot become too negative. But *INV* would not hold any longer if we could start with arbitrary integer values of z initially. This is exactly the point where our approach differs from that in [3].

There the priority variables are always non-negative. Thus there is no need for initialization and no need for the invariant *INV* either. However, the transformations used to derive and justify the proof rules in [3] are not *faithful* in the sense of Section 3 any more because they allow additional failures caused by if-fi constructs where all guards are false. It is interesting to note that these failures can be absorbed nicely in the proof rules of [3]. On the other hand, it turns out that the invariants used in the formal proof are quite robust against such changes of the underlying transformations. This was also our experience when we simplified the transformations used in the earlier version [1]: the proofs remained valid almost unchanged. This observation is rather reassuring. It shows that when applying our proof rules we are really analyzing the structure of the problem considered and we are not fighting with some obscure proof rules.

Another approach to the problem of proving total correctness under the assumption of fairness was (independently) developed in [11] and [10]. The idea of this approach is to relax the usual requirement for providing total correctness [9], namely that *every* action (i.e. execution of a component S_i of the do-od-loop) causes a decrease of a well-founded quantity α. Instead actions are divided into *helpful* ones which always decrease α and the other ones—called *indifferent* or *steady*—which do not increase α. By fairness and some additional requirements of the method a helpful action must eventually be chosen which causes to decrease and thus excludes infinite computations. This method was applied in [11] to concurrent programs represented in an abstract framework, and in [10] to the class of program considered in our paper.

Even though this method and ours are based on a different intuition they should prove equivalent. This follows of course from the fact that both methods are sound and complete. But there should also exist a direct translation of invariant for one method into an invariant for the other method. We did not check the details but such a translation should take into account the intuition that priority variables count the number of times an action will be enabled but not yet chosen.

A unification of these methods is already attempted in [3] where the underlying ideas are formalized in both frameworks—that of delay variables and that of helpful and steady actions.

In our future work we intend to extend the approach presented here to various concurrent languages.

Acknowleldgment

We are indebted to Amir Pnueli for his very helpful and detailed referee reports. Research on this paper started in October 1980 in Rotterdam and was subsequently continued by letter and during meetings in Kiel, Bad Honnef, Peniscola, Akko, Garmisch-Partenkirchen, Amsterdam, Paris and Oxford. We thank all those who provided support for these meetings.

References

[1] K.R. Apt and E.-R. Olderog, Proof rules dealing with fairness (extended abstract), *Proc. Logics of Programs*, Lecture Notes in Computer Science **131** (Springer, Berlin, 1982) 1–8.

[2] K.R. Apt and G.D. Plotkin, Countable nondeterminism and random assignment, Technical Report, University of Edinburgh, 1982 (extended abstract appeared in: *Proc. ICALP 81*, Lecture Notes in Computer Science **115** (Springer, Berlin, 1981) 479–494).

[3] K.R. Apt, A. Pnueli and J. Stavi, Fair termination revisited—with delay, *Proc. 2nd Conference on Software Technology and Theoretical Computer Science*, Bangalore, India (1982) 146–170.

[4] P. Cousot, Asynchronous iterative methods for solving a fixed point system of monotone equations in a complete lattice, Rapport de Recherche No 88, L.A.7, Université Scientifique et Medicale de Grenoble (1977).

[5] E.W. Dijkstra, *A Discipline of Programming* (Prentice-Hall, Englewood Cliffs, 1976).

[6] E.A. Emerson and E.M. Clarke, Characterizing correctness properties of parallel programs using fixpoints, *Proc. ICALP 80*, Lecture Notes in Computer Science **85** (Springer, Berlin, 1980) 169–181.
[6] M.J. Fischer and M.S. Paterson, Storage requirements for fair scheduling, Note (1982).
[7] L. Flon and N. Suzuki, Nondeterminism and the correctness of parallel programs, *Proc. IFIP TC-2 Working Conference in Formal Description of Programming Concepts* (North-Holland, Amsterdam, 1978) 589–608.
[8] L. Flon and N. Suzuki, The total correctness of parallel programs, *SIAM J. Comput.* **10** (2) (1981) 227–246.
[9] R.W. Floyd, Assigning meaning to programs, *Proc. Symposium in Applied Mathematics, Vol. 19, Mathematical Aspects of Computer Science* (American Mathematical Society, Providence, RI, 1967) 19–32.
[10] O. Grümberg, N. Francez, J.A. Makowsky and W.P. de Roever, A proof rule for fair termination of guarded commands, *Proc. International Symposium on Algorithmic Languages* (North-Holland, Amsterdam, 1981) 339–416.
[11] D. Lehmann, A. Pnueli and J. Stavi, Impartiality, justice and fairness: the ethics of concurrent termination, *Proc. ICALP 81*, Lecture Notes in Computer Science **115** (Springer, Berlin, 1981) 264–277.
[12] D. Park, A predicate transformer for weak fair iteration, *Proc. IBM Symposium on Mathematical Foundation of Computer Science*, Hakone (1981).
[13] A. Pnueli, The temporal semantics of concurrent programs, *Theoret. Comput. Sci.* **13** (1981) 45–60.