Progress Measures and Stack Assertions for Fair Termination

Nils Klarlund*
IBM T.J. Watson Research Center
PO BOX 704
Yorktown Heights, New York

Abstract

Floyd's method based on well-orderings is the standard approach to proving termination of programs. Much attention has been devoted to generalizing this method to termination of programs that are subjected to fairness constraints. Earlier methods for fair termination tend to be somewhat indirect, relying on program transformations, which reduce the original problem to several termination problems.

In this paper we introduce the new concept of stack assertions, which directly—without transformations—quantify progress towards fair termination. Moreover, we show that by one simple program transformation of adding a history variable, usual assertional logic, without fixed-point operators, is sufficiently expressive to form a sound and relatively complete method when used with stack assertions. This result is obtained as part of a substantial simplification of earlier completeness proofs.

1 Introduction

Fairness is the assumption that an action that is enabled over and over will eventually be taken. Such assumptions are central to many distributed or concurrent systems. The fair termination problem—how to prove that a program terminates under assumption of fairness—is typical to much reasoning with fairness, and many methods for this problem have been suggested; see [AO83, AFK88, DH86, FK84, Fra86, GFMdRv85, LPS81, MP91, SdRG89]. Most of the methods build on Floyd's approach of using well-ordered sets as a measure of how close the program is to termination. Floyd's ideas allow one to annotate the unaltered program with assertions expressing closeness to termination, whereas many of the earlier methods for fair termination depend on changing the program. The modifications either consist of adding new program variables and unbounded nondeterminism, or involve recursively applied proof rules that transform the program. Unfortunately, these modifications tend to obscure how each step of the original program contributes to fair termination.

The goal of this paper is a lucid and practical approach for showing fair termination without repeated or drastic transformations. Our approach is based on the novel concept of *progress measure* introduced in [Kla90]; see also [Kla, KK91, Kla91, KS93]. A progress measure is a function on the states (or histories) of the *unaltered* program. The value of the function for a given state quantifies—in a certain mathematical sense—how close that state is to satisfying a property about infinite computations. The property is defined by a *specification*, which characterizes states (or histories), and by a *limit condition*, which when applied to the specification defines the allowed infinite computations.

The property, for example, could be that every infinite computation is unfair—this means that the program fairly terminates. In this case, the specification characterizes for each state which actions are enabled and which are taken; the limit condition expresses the fixed temporal meaning of unfairness: some action is enabled infinitely often while being taken only finitely often.

An essential property of a progress measure is that on every program transition, its value changes in a way ensuring that the computation converges according to the limit condition. This requirement can be formulated as *verification conditions*, which allow verification of *global* properties (on infinite computations) in terms of *local* reasoning (about states and transitions).

*This work was mainly carried out while the author was with the IBM T.J. Watson Research Center. The work has also been supported by an Alice & Richard Netter Scholarship of the Thanks to Scandinavia Foundation, Inc.; Forskerakademiet, Denmark; and Esprit Basic Research Action Grant No. 3011, Cedisys. Author's current address: Aarhus University, Department of Computer Science, Ny Munkegade, DK-8000 Aarhus, Denmark. E-mail: klarlund@daimi.aau.dk.

PoDC '92-8/92/B.C.
© 1992 ACM 0-89791-496-1/92/0008/0229...$1.50

This correspondence is especially meaningful if the local reasoning can be done for an unmodified program, in which case it becomes clear what the exact contribution of each program step (by each process) is towards the global behavior. In this paper we provide a practical tool, called a *stack assertion*, that provides such an understanding for distributed or concurrent programs that fairly terminate. The stack assertions of a program define a mapping, called a *fair termination measure*, that describes how close each program state is to fair termination.

The contributions of this paper are both practical and theoretical. We demonstrate the usefulness of stack assertions by examples. For distributed or concurrent programs, our examples indicate a direct way of contributing "lack of progress towards termination" to "progress towards unfair execution" as expressed by a hierarchy of unfairness hypotheses. Stack assertions form the natural framework for expressing this hierarchy and summarize in a single data structure the information obtained by the program transformations of previous methods. Since the need for transformations has been eliminated, stack assertions can be added to existing assertional methods for concurrent and distributed programs.

There are two theoretical results of this paper. The first is a new completeness proof—substantially simpler than earlier proofs that involve transfinite induction or results from topology—that explains why a fair termination measure always exists for programs or distributed systems that fairly terminate.

The second result is that by adding a history variable to a program, the fair termination measure can be expressed by means of stack assertions in any reasonably expressive assertion language (i.e. a language that includes arithmetic).

In some earlier work on expressing assertions about fair termination [SdRG89, Mor90, MP91], predicate calculus is combined with fixed-points and ordinals. For an arbitrary program, this calculus allows to characterize precisely the states from which all infinite computations are unfair.

In the present paper we show that with the addition of a history variable, an assertion language containing only predicate calculus is sufficient for a proof of fair termination from the initial state. In our method, well-foundedness is expressed not by fixed-point logic in program assertions, but as an additional requirement that a relation, expressed by the program assertions, is well-founded (has no infinite descending chains.) Observe that adding history information (as for example is also done in methods for verification with nondeterministic automata [AL91, Sis91]) is a benign transformation in that it has to be done only once and basically does not change the transitional structure of the program; in particular, no additional nondeterminism is added.

2 Verification Methods for Fairness

A *fairness constraint* partitions infinite computations into fair and unfair ones. In this paper we shall concentrate on *strong fairness*, which is one of the most important fairness concepts. According to this criterion, a computation is *fair* if commands (or processes, statements, actions, events,...) that are enabled infinitely often are also executed infinitely often. (It is assumed that the number of different commands is finite.) Thus an *unfair* computation is one where some command is enabled infinitely often but only executed finitely often. A program P *fairly terminates* if every infinite computation of P is unfair. A *verification method* for fair termination is defined in terms of verification conditions expressed in the style of Hoare's logic. To be useful a method must be *sound*, i.e. any program for which the verification conditions can be satisfied must fairly terminate. The method is *complete* if the verification conditions can be satisfied for any program that fairly terminates.

Complete verification methods for strongly fair termination are considered in [GFMdRv85, LPS81, SdRG89]. These methods are based on *helpful directions*, which indicate program statements that are being unfairly executed. The approach of helpful directions has been successful at explaining many fairness concepts, such as those involving general state predicates [FK84] or an infinite number of commands [Mai89]. All these methods involve the recursive use of proof rules that are applied to transformed programs. Thus they tend to depend on particular syntactic properties of the underlying program language [Fra86, page 117] (although a way of circumventing these syntactic dependencies is indicated in [Fra86, section 2.4]).

The methods of *explicit schedulers* developed in [AO83, APS84, DH86] involve transforming programs by adding auxiliary variables that are nondeterministically assigned values determining fair computations. Because they involve rather drastic—even "cruel" [DH86]—program transformations, these methods also deal with fairness in a somewhat indirect manner. For an extensive treatment of fairness based on helpful directions and explicit schedulers, see the book [Fra86].

In [SdRG89] it was shown how predicate calculus augmented with fixed-points can be used to express assertions about fair termination. This calculus can express *inductively definable relations* [Mos74], which

are needed in their proof.[1] Usually, however, assertional reasoning is based on ordinary predicate calculus, which corresponds to *arithmetic relations* [Rog67]. Earlier, Apt and Plotkin, motivated by the relationship mentioned above between fairness and nondeterminism, gave a semantic model for countable nondeterminism [AP86]. In addition, they provided a relatively complete proof system for termination, also based on fixed-point logic.

Using a fragment of fixed-point calculus, Manna and Pnueli formulated elegant proof rules for assertional reasoning about properties expressed in temporal logic. For the problem of fair response (which generalizes fair termination), they exhibited a simple proof rule, which is recursively applied to transformed programs.

Morris [Mor90] also used fixed-point calculus in his formulation of a weakest precondition semantics for fair termination of tail-recursive programs.

The work presented here is also related to the theory of automata on infinite words. In fact, the condition of fair termination is but an instance of a *Rabin pairs condition*, see [KK91], which is a requirement in a special disjunctive normal form about the infinite occurrence of states. The proofs in the present paper could have been formulated for Rabin pairs conditions (thus yielding a method for *general fairness* [FK84]), but for simplicity of exposition we have used conditions pertaining to strong fairness.

The *Rabin progress measures* in [KK91] express progress towards satisfaction of a Rabin pairs conditions. Applied to fair termination, a Rabin progress measure maps program states into a special kind of colored trees. This gives a concise method for fair termination that does not depend on program transformations [KK91]. The method is not entirely practical, however, because there is no natural way to describe the mapping into the colored tree, which has to be described explicitly—obstacles that are overcome in this paper. For a more detailed comparison, see Section 5.

A concept similar to our stacks is used in [Saf92], where the problem is to determinize an automaton with a *Streett condition* (a special conjunction) or, equivalently, to express that a Rabin condition holds along all computations of a nondeterministic automaton by means of a deterministic automaton. For complementation of tree automata, the *last appearance record* of [GH82] serves a purpose different from that of our stacks, namely to keep enough information about the past to make finitely represented choices in a winning strategy for a game that is won by satisfying a conjunction (such as a Streett condition). On the other hand, stacks in the form of Rabin progress measures can be used to show that there is no need for such information when the game is won by a disjunction (Rabin condition) [Kla92].

Finally, our work is related to more general techniques for proving liveness properties. Harel showed that by transformations on trees representing programs one can do program verification for all finite levels of the Borel hierarchy [Har86]. Using an automata-theoretic approach, Vardi gave a verification method for very general properties, including the Borel hierarchy [Var87]. In Vardi's framework, progress is measured relative to a nondeterministic automaton that defines incorrect computations. In contrast, the *progress measures* of [Kla90, Kla] are functions that relate the program state or history to a finite computation of a correctness specification. With this approach nondeterminism must be eliminated, since it makes it difficult to relate program to specification by means of a function (cf. the work [AL91, KS93, Sis91] on relating automata defining safety properties). Instead more powerful limit conditions than those usually studied (e.g. Rabin or Streett conditions) are used to define the infinite computations as limits of finite ones.

3 Stack Assertions

In this section we review the method of Floyd and explain how assertions can define a measure of progress for termination. We argue informally how stack assertions can be used to guarantee fair termination. For simplicity we present our examples using the language of guarded commands, but our technique is syntax-independent and also applies to strong fairness expressed for other formalisms describing distributed or concurrent systems.

3.1 Floyd's Method

For programs occurring in practice it is usually straightforward to quantify progress towards termination. This is done in terms of well-founded sets as first advocated by Floyd [Flo67]. A *well-founded set* $(W, \succ)$ is a set W with a binary relation $\succ$ such that there is no infinite descending sequence $w_0 \succ w_1 \succ \cdots$. For an example of proving termination, take the program

$$P1 : *[\ x < y \ \rightarrow x := x + 1\ ]$$

[1] The inductively definable relations are the same as the Π^1_1 relations. Π^1_1 is the class of relations on the form $\forall\alpha : p$, where α is a second-order object (such as an infinite computation) and p is a first-order formula (such as the one expressing that a computation is unfair). The problem of fair termination is Π^1_1-complete as is the problem of termination (of programs with countable nondeterminism).

consisting of a loop with a single guarded command, which is executed as long as its guard, $x < y$, is enabled (true). The variables take on integer values. To argue that $P1$ terminates, we use the mapping $\mu^{\mathrm{T}} = \max\{y-x, 0\}$ from program states into the well-founded set of natural numbers $0 < 1 < 2 < \cdots$. Here and in the sequel, the letter "T" refers to the hypothesis that the programs terminates. The mapping μ^{T} can be called a *termination measure*, since its value decreases with each iteration. The existence of a termination measure μ^{T} guarantees that P terminates, because an infinite computation $p_0, p_1, \ldots$ would produce an infinite descending sequence $\mu^{\mathrm{T}}(p_0) > \mu^{\mathrm{T}}(p_1) > \cdots$, contradicting the well-foundedness of the natural numbers.

In practice, the termination measure μ^{T} is expressed by annotating the program with assertions. For $P1$ a single assertion suffices:

$$P1'\colon *[\ \left(\mathrm{T} : \max\{y-x, 0\}\right) \quad x < y \ \rightarrow x := x+1\ ]$$

Here $\left(\mathrm{T} : \max\{y-x, 0\}\right)$ is a simple *stack assertion*. It asserts that for the termination hypothesis, also called the T-*hypothesis*, the value of the termination measure μ^{T} is $\max\{y-x, 0\}$ whenever the loop is to be executed. Thus it could be called a *loop variant*[2] as opposed to a *loop invariant*. The latter expresses a relationship between variables that is preserved under iterations.

3.2 Fair Termination

With a trivial modification of $P1$, proving termination is suddenly more intricate. Consider the program

$$P2\colon *[\ \ell_a\colon x < y \ \rightarrow x := x+1\ \square\ \ell_b\colon x < y \ \rightarrow \texttt{skip}\ ]$$

where the loop is executed as long as $x < y$ by execution of either of the guarded commands $x < y \rightarrow x := x+1$ and $x < y \rightarrow \texttt{skip}$. The choice is made nondeterministically. This program will not terminate if the second command is always chosen from some point on. Under assumption of (strong) fairness, however, $P2$ always terminates, because in an infinite computation of $P2$, ℓ_a is only executed finitely often, but enabled infinitely often; thus the computation is unfair with respect to command ℓ_a.

[2]The terms *variant function* or *bound function* are also used [Gri81].

The preceding argument was formulated in terms of infinite computations. In contrast, assertional reasoning deals only with program states and single transitions. The key to assertional reasoning about fairness is:

> If there is no progress towards termination, this can be attributed to some statement being executed unfairly.

For example, when ℓ_b is executed, the T-hypothesis is not active since there is no progress towards termination. Instead, progress towards executing ℓ_a *unfairly* can be measured. To do this we reformulate the assertion by including the *unfairness hypothesis*, called the ℓ_a-*hypothesis*, that ℓ_a is executed unfairly. Syntactically, this is done by putting "ℓ_a" on top of the underlying T-hypothesis; thus we write

$$\left(\frac{\ell_a}{\mathrm{T} : \max\{y-x, 0\}}\right).$$

This stack assertion expresses a hierarchy in which the T-hypothesis is the underlying hypothesis and the rôle of the ℓ_a-hypothesis is to explain progress when the underlying hypothesis can not. The annotated program is now:

$$P2'\colon *[\ \left(\frac{\ell_a}{\mathrm{T} : \max\{y-x, 0\}}\right) \quad \ell_a\colon x < y \ \rightarrow x := x+1\ \square\ \ell_b\colon x < y \ \rightarrow \texttt{skip}\ ]$$

Progress is made towards unfair execution in terms of the ℓ_a-hypothesis whenever ℓ_a is enabled but not executed. Note that for any iteration, either

- ($\mathrm{V_a}$) ℓ_a is enabled and not executed, and the underlying T-measure remains constant (when ℓ_b is executed); or
- ($\mathrm{V_T}$) measure μ^{T} decreases (when ℓ_a is executed).

We can now argue that $P2$ fairly terminates in terms of the local conditions ($\mathrm{V_a}$) and ($\mathrm{V_T}$) as follows. In an infinite computation, either from some point on ($\mathrm{V_a}$) always applies, or infinitely often ($\mathrm{V_T}$) applies. In the first case, ℓ_a is always enabled but never executed. Hence the computation is unfair with respect to ℓ_a. In the second case, it holds that each time ($\mathrm{V_T}$) applies, μ^{T} is decreased, and at the other times, when ($\mathrm{V_a}$) applies, μ^{T} is unchanged. This yields an infinite decreasing sequence of natural numbers, which is a contradiction.

Thus we have proved that for any infinite computation of $P2$, only the first case is possible, i.e. $P2$ fairly terminates. This argument will later be generalized to a

soundness result, which shows that a verification condition, similar to the local conditions above, always implies fair termination. For now, however, we motivate this general result by looking at more examples.

3.3 Progress Measures for Unfairness Hypotheses

A more complex situation arises if ℓ_a is sometimes not enabled when ℓ_b is executed. In this case the stack assertion $\left(\frac{\ell_a}{\mathrm{T} : \max\{y-x,0\}}\right)$ cannot be applied, because condition $(\mathrm{V_a})$ is then sometimes not fulfilled. If the program fairly terminates, however, we can use a progress measure μ^{ℓ_a} for the ℓ_a-hypothesis. For an example of this situation, take:

$$
\begin{array}{lll}
P3\colon *[\ \ell_a\colon x < y \wedge z \bmod 117 = 0 & \rightarrow x := x+1 & \square \\
\quad\ \ell_b\colon x < y & \rightarrow z := z-1\ ] &
\end{array}
$$

This program fairly terminates, because for any infinite computation, ℓ_a can only be executed finitely often and the value of z decreases by one each time ℓ_b is executed; thus ℓ_a is enabled infinitely often. We might annotate the program as follows:

$$
\begin{array}{lll}
P3'\colon *[\ \left(\frac{\ell_a\ :\ z \bmod 117}{\mathrm{T} : \max\{y-x,0\}}\right) & & \\
\quad \ell_a\colon x < y \wedge z \bmod 117 = 0 & \rightarrow x := x+1 & \square \\
\quad \ell_b\colon x < y & \rightarrow z := z-1\ ] &
\end{array}
$$

Here $a : z \bmod 117$ denotes that a progress measure $\mu^{\ell_a} = z \bmod 117$ is associated with the ℓ_a-hypothesis. The measure μ^{ℓ_a} is a measure of how close $P3$ is to a state in which ℓ_a is enabled. For each iteration of the loop, either

- $(\mathrm{V'_a})$ measure μ^{T} is unchanged, ℓ_a is not executed, and either the value of z was 0 (mod 117) before the execution of ℓ_b, in which case ℓ_a was enabled, or the value of z (mod 117) was between 1 and 116 and decreases by 1; or
- $(\mathrm{V'_T})$ measure μ^{T} decreases.

The local conditions $(\mathrm{V'_a})$ and $(\mathrm{V'_T})$ ensure that an infinite computation is unfair with respect to ℓ_a. Consider the corresponding infinite sequence of stacks. It must be the case that from some point on, $(\mathrm{V'_a})$ applies to each transition. Thus ℓ_a is only executed finitely often. If from some point on it is never enabled, then μ^{ℓ_a} decreases for each iteration thereafter, contradicting the well-foundedness of the natural numbers. Therefore, ℓ_a is enabled infinitely often, and we conclude that any infinite computation is unfair with respect to ℓ_a.

3.4 Unfairness of Several Commands

An even more challenging situation occurs when more than one command may be executed unfairly. If we add an empty guarded command to $P3$, we obtain:

$$
\begin{array}{lll}
P4\colon *[\ \ell_a\colon x < y \wedge z \bmod 117 = 0 & \rightarrow x := x+1 & \square \\
\quad\ \ell_b\colon x < y & \rightarrow z := z-1 & \square \\
\quad\ \ell_c\colon x < y & \rightarrow \texttt{skip}\ ] &
\end{array}
$$

This program fairly terminates, because any infinite computation is unfair with respect to either ℓ_a or ℓ_b. To see this we use the loop variant from $P3'$ modified to explain the lack of progress when ℓ_c is chosen for execution. In that case there is progress neither towards termination nor towards executing ℓ_a unfairly. But there *is* always progress towards something when a program fairly terminates; in fact, when ℓ_c is executed, ℓ_b is a candidate for unfair execution because it is enabled but not executed. Thus we can put the ℓ_b-hypothesis—that ℓ_b is executed unfairly—on top of the T- and ℓ_a-hypotheses. The annotated program then becomes:

$$
\begin{array}{lll}
P4'\colon *[\ \left(\frac{\ell_b}{\frac{\ell_a\ :\ z \bmod 117}{\mathrm{T} : \max\{y-x,0\}}}\right) & & \\
\quad \ell_a\colon x < y \wedge z \bmod 117 = 0 & \rightarrow x := x+1 & \square \\
\quad \ell_b\colon x < y & \rightarrow z := z-1 & \square \\
\quad \ell_c\colon x < y & \rightarrow \texttt{skip}\ ] &
\end{array}
$$

This annotation can be used as an argument why $P4$ fairly terminates in a way similar to the previous arguments.

Note that if earlier methods involving recursive proof rules had been used instead to show that $P4$ fairly terminates, it would have been necessary to reason about three different programs: the original and two syntactically derived programs.

4 Verification Conditions, Soundness, and Completeness

In the preceding section we developed a notation for reasoning about fairness. For each program we considered an arbitrary infinite computation and argued, using the associated stacks, that the computation was unfair. The rationale for using stack assertions, however, is to avoid reasoning about infinite computations. So to obtain an assertional verification method, we formulate verification conditions for the stacks expressed by the assertions. When these conditions are fulfilled for all transitions, we say that the stack assertions define a *fair termination measure*. We show

that if a program has a fair termination measure, then it fairly terminates. Thus the verification method of stack assertions is sound.

We also give a completeness result: when a program fairly terminates, it has a fair termination measure. Moreover, we show that under certain conditions (which are fulfilled if a history variable is added to the program) then the fair termination measure can be expressed as program assertions in a reasonably powerful assertion language.

4.1 The Verification Conditions

To formulate the verification conditions we need a few definitions. A *program* P defines a transition relation $\rightarrow$ on a countable set of program states; moreover, P defines a set of initial program states and a finite set of commands. A *command* (or an action, a process, an event,...) is designated by a label ℓ, and P defines for each program state whether ℓ is enabled or disabled. A *transition* $p \rightarrow p'$ describes the execution of exactly one command, which is enabled in p. A *path* from p_0 to p_n is a sequence of states $p_0, \ldots, p_n$ such that $p_i \rightarrow p_{i+1}$ for $i < n$; an infinite path $p_0, p_1, \ldots$ is defined in a similar way. A *computation* is a finite or infinite path starting in an initial state. A state p' is *reachable* from state p if there is a path from p to p'. We assume without loss of generality that any program state is reachable from an initial state. (In practice, conventional assertional methods can be used to describe the reachable program states, since finite sequences of program states can be encoded as numbers; see [MP91].)

A *progress hypothesis* or α-hypothesis is either an unfairness hypothesis, on the form ℓ or $\ell : w$ (with $\alpha = \ell$), or the T-hypothesis, on the form $T : w$ (with $\alpha = \ell$), where w is an element of a well-founded set $(W, \succ)$. A *stack assignment* is a mapping that maps each program state p to a list $\mu(p)$ of progress hypotheses such that the T-hypothesis is at level 0, i.e. at the bottom. (It can be assumed that all the hypotheses are different, i.e. there is at most one ℓ-hypothesis in $\mu(p)$ for each ℓ.) The stack assertions of a program define a stack assignment according to the semantics of the logical language of the assertions. (The exact correspondence is of no importance here.) For an hypothesis $\alpha : w$ in $\mu(p)$, where α is a label or "T," the value w is called the *α-measure* at p and is denoted $\mu^\alpha(p)$.

Note that the definitions above are not dependent on the particular syntax of guarded commands, but depend only on the notions of commands or actions being "enabled" and "executed." Thus our soundness and completeness results apply to strong fairness in all transition systems. For example, our method applies to nested commands ("all-level fairness," see [Fra86]).

The verification conditions are expressed in terms of active and non-invalidated hypotheses: essentially, an ℓ-hypothesis is *active* if progress towards unfair execution of ℓ is made, and it is *non-invalidated* if ℓ is not executed. The T-hypothesis is active if the program gets closer to termination; the T-hypothesis is always considered non-invalidated.

The verification conditions can now be stated somewhat informally:

(V_F) On any program transition,
- there is some active hypothesis;
- the active hypothesis and the ones below are non-invalidated;
- and the stack does not change below the active hypothesis.

The meaning of this is illustrated in Figure 1. Here the program transition is $p \rightarrow p'$. The active hypothesis α is at the same level in the stacks $\mu(p)$ and $\mu(p')$, and everything below (denoted by S in the figure) remains unchanged. Formally, the verification conditions (V_F) are:

(V_A) Some α-hypothesis is *active*, i.e. either
- α is a label ℓ and command ℓ is enabled (in state p or p'), or
- $w = \mu^\alpha(p)$ and $w' = \mu^\alpha(p')$ are defined with $w \succ w'$.

(V_{NonI}) Every hypothesis below and including the α-hypothesis is *non-invalidated* i.e. none of these hypotheses is the ℓ-hypothesis, where ℓ is the command executed in going from p to p'.

(V_{NoC}) The stack does not change below hypothesis α.

The contents of the stack above α may change in any way. When the stack assignment μ satisfies these conditions for all program transitions, we say that $(\mu, (W, \succ))$—or μ (when the well-founded relation $(W, \succ)$ is understood from the context)—is a *fair termination measure*.

4.2 Example

Here is an argument explaining why the stack assertion of $P4'$ satisfies (V_F). Consider an iteration not leading to termination. There are three cases depending on which command is executed:

ℓ_a: The T-hypothesis is active, because $\mu^T = \max\{y - x, 0\}$ decreases (since $x < y$ holds before ℓ_a is executed). There is nothing beneath the T-hypothesis to check.

ℓ_b: Below the ℓ_a-hypothesis, the stack remains unchanged and the T-hypothesis is not invalidated.

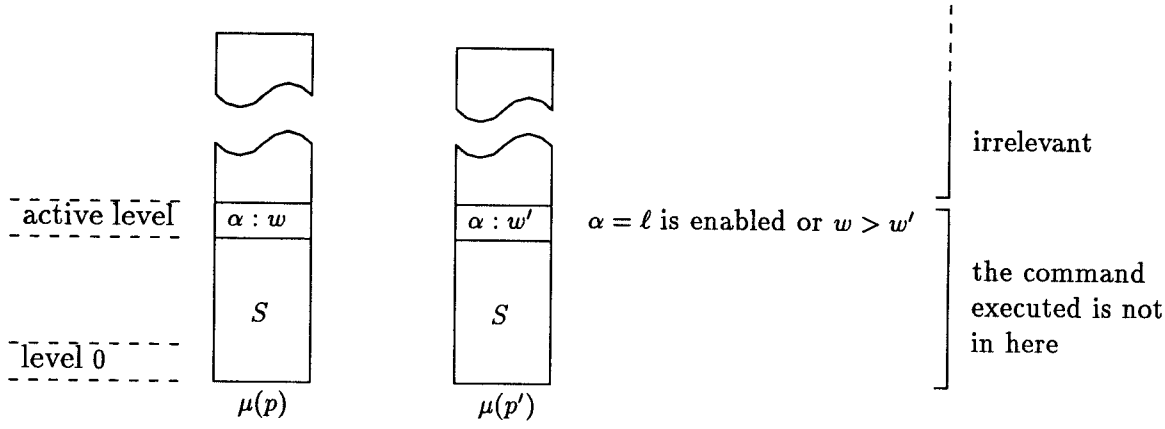

Figure 1: Verification condition.

The ℓ_a-hypothesis is active, because ℓ_a is not executed, and before execution of ℓ_b, either $z = 0 \pmod{117}$ holds, i.e. ℓ_a is enabled, or $z \neq 0 \pmod{117}$ holds, i.e. $\mu^{\ell_a} = z \bmod 117$ decreases.

ℓ_c: The stack is unchanged below the ℓ_b-hypothesis. The ℓ_b-hypothesis is active, because ℓ_b is enabled but not executed. The ℓ_a-hypothesis is non-invalidated, because ℓ_a is not executed.

4.3 Soundness

Theorem 1 (Soundness of Fair Termination Measures) If P has a fair termination measure, then P fairly terminates.

(See the Appendix for all proofs.)

4.4 Completeness

Theorem 2 (Completeness of Fair Termination Measures) If P fairly terminates, then P has a fair termination measure.

To prove Theorem 2, we first present a simple completeness proof, which applies to programs that are tree-like. A program is *tree-like* if it has a single initial state p^0 and if every state p', except p^0, has exactly one predecessor, i.e. there is exactly one p such that there is a transition $p \rightarrow p'$. Any program can be made tree-like by adding a history variable recording the past sequence of program states.

Theorem 3 If P fairly terminates and is tree-like, then P has a fair termination measure.

To prove Theorem 2 for an arbitrary program P, we apply Theorem 3 to the tree-like program P' that is obtained by adding a history variable to P. The value of the progress measure for a state p of P is then chosen as the least value of the progress measure of states in P' that correspond to p; here "least" means least with respect to a cross-product ordering on the progress measures of P'.

4.5 Relative Completeness

It is not hard to see that the completeness result in Theorem 3 can be sharpened to show that an effectively represented fair termination measure exists for an *effectively represented program* P (a program that has a recursive transition relation[3] and a recursive function that for all p' defines the state p (if it exists) such that $p \rightarrow p'$.) In fact, this measure can be obtained uniformly from P. To see this we define a *fair termination semi-measure* $(\mu, (W, \succ))$ to be a fair termination measure *except* that W need not be well-founded; thus μ is just required to satisfy the verification conditions.

Theorem 4 There is a recursive function h that given an index for a tree-like program P gives indices for a fair termination semi-measure $(\mu, (W, \succ))$, where both μ and $(W, \succ)$ are recursive. Moreover, $(\mu, (W, \succ))$ is a fair termination measure (i.e. $(W, \succ)$ is well-founded) iff P is fairly terminating.

This theorem gives an explicit reduction of the fair termination problem to a classical Π^1_1-complete problem of whether a recursive relation is well-founded. Moreover, it shows that if the assertional language includes usual predicate logic on numbers (and therefore all relations in the arithmetic hierarchy by a fundamental result of Gödel, see [Rog67]), then there exists a stack assertion

$$\left(\begin{array}{c} \alpha_N : w_N \\ \cdots \\ \hline \alpha_1 : w_1 \\ \hline \mathrm{T} : w_T \end{array}\right),$$

[3] A *recursive relation* is also called a recursively computable relation, see [Rog67].

where the α's and w's are definable in the assertional logic, that satisfies the verification conditions.

Thus we obtain:

Corollary 1 (Relative Completeness of Stack Assertions) If the assertional language contains predicate calculus and if P fairly terminates, then P can be annotated with stack assertions in terms of the program history such that the verification conditions are satisfied.

5 Discussion

The results presented here are related to the method of helpful directions [Fra86, GFMdRv85, LPS81] and the Rabin measures of [KK91].

Formulated in our terminology, the method of helpful directions is used to identify one level of the fair termination measure at a time. For example, one first identifies subsets of program states corresponding to a constant μ^{T} measure. Then the program is transformed into several new programs, each corresponding to a subset. The states of each derived program are then further partitioned according to unfairness hypothesis (helpful directions) of the first level to yield more subsets, which are expressed as more derived programs.

Our approach is also related to the Rabin progress measures of [KK91, Kla90]. A Rabin progress measure is defined as a mapping from the program states into a colored tree. This mapping can be described in program assertions by specifying the progress values for each program state. The problem is that the colored tree has to be explicitly described (as it was done in an example given in [KK91]). In contrast, the stack assertions given in this paper are self-contained.

There are some technical differences that have been introduced to make stack assertions more useful for program annotation:

- Two stacks may contain the same progress values, but be colored differently. In a Rabin progress measure the coloring is a function of the progress values. Thus it is not possible to translate directly a fair termination measure into a Rabin progress measure.
- For a Rabin progress measure, satisfaction of an enabling condition is expressed in terms of the new state. For stack assertions, the satisfaction of the enabling condition is considered in terms of the old state and the new state.
- There may be several choices for an active hypothesis. For Rabin progress measures the active hypothesis is uniquely determined for each transition.

Acknowledgements

Thanks to Martín Abadi for very helpful comments. Dexter Kozen, Fred B. Schneider, and Mike Slifker also provided valuable comments on earlier versions of this paper.

References

[AFK88] K.R. Apt, N. Francez, and S. Katz. Appraising fairness in languages for distributed programming. *Distributed Computing*, 2:226–241, 1988.

[AL91] M. Abadi and L. Lamport. The existence of refinement mappings. *Theoretical Computer Science*, 82(2):253–284, 1991.

[AO83] K.R. Apt and E.-R. Olderog. Proof rules and transformations dealing with fairness. *Science of Computer Programming*, 3:65–100, 1983.

[AP86] K.R. Apt and G.D. Plotkin. Countable nondeterminism and random assignment. *JACM*, 33(4):724–767, 1986.

[APS84] K.R. Apt, A. Pnueli, and J. Stavi. Fair termination revisited with delay. *Theoretical Computer Science*, 33:65–84, 1984.

[DH86] I. Dayan and D. Harel. Fair termination with cruel schedulers. *Fundamenta Informatica*, 9:1–12, 1986.

[FK84] N. Francez and D. Kozen. Generalized fair termination. In *Proc. 11th POPL, Salt Lake City.* ACM, January 1984.

[Flo67] R. Floyd. Assigning meaning to programs. In *Mathematical Aspects of Computer Science XIX*, pages 19–32. American Mathematical Society, 1967.

[Fra86] Nissim Francez. *Fairness.* Springer-Verlag, 1986.

[GFMdRv85] O. Grumberg, N. Francez, J.A. Makowsky, and W.P. de Roever. A proof rule for fair termination of guarded commands. *Information and Control*, 66(1/2):83–102, 1985.

[GH82] Y. Gurevich and L. Harrington. Trees, automata, and games. In *Proceedings 14th Symp. on Theory of Computing.* ACM, 1982.

[Gri81] David Gries. *The Science Of Programming.* Springer-Verlag, 1981.

[Har86] D. Harel. Effective transformations on infinite trees with applications to high undecidability, dominos, and fairness. *Journal of the ACM*, 33(1):224–248, 1986.

[KK91] N. Klarlund and D. Kozen. Rabin measures and their applications to fairness and automata theory. In *Proc. Sixth Symp. on Logic in Computer Science*. IEEE, 1991.

[Kla] N. Klarlund. Liminf progress measures. In *Proc. of Mathematical Foundations of Programming Semantics 1991*. To appear in LNCS.

[Kla90] Nils Klarlund. *Progress Measures and Finite Arguments for Infinite Computations*. PhD thesis, TR-1153, Cornell University, August 1990.

[Kla91] N. Klarlund. Progress measures for complementation of ω-automata with applications to temporal logic. In *Proc. Foundations of Computer Science*. IEEE, 1991.

[Kla92] N. Klarlund. Progress measures, immediate determinacy, and a subset construction for tree automata. In *Proc. Seventh Symp. on Logic in Computer Science*, 1992. To appear.

[KS93] N. Klarlund and F.B. Schneider. Proving nondeterministically specified safety properties using progress measures. To appear in *Information and Computation*, 1993.

[LPS81] D. Lehmann, A. Pnueli, and J. Stavi. Impartiality, justice and fairness: the ethics of concurrent termination. In *Proc. 8th ICALP*. LNCS 115, Springer-Verlag, 1981.

[Mai89] M.G. Main. Complete proof rules for strong fairness and strong extreme-fairness. Technical Report CU-CS-447-89, Department of Computer Science, University of Colorado, 1989.

[Mor90] J.M. Morris. Temporal predicate transformers and fair termination. *Acta Informatica*, 27:287–313, 1990.

[Mos74] Y.N. Moschovakis. *Elementary Induction on Abstract Structures*. North-Holland, 1974.

[MP91] Z. Manna and A. Pnueli. Completing the temporal picture. *Theoretical Computer Science*, 83:97–103, 1991.

[Rog67] Hartley Rogers, Jr. *Theory of Recursive Functions and Effective Computability*. McGraw-Hill Book Company, 1967.

[Saf92] S. Safra. Exponential determinization for ω-automata with strong-fairness acceptance condition. In *Proc. 24th Symposium on Theory of Computing*, 1992.

[SdRG89] F.A. Stomp, W.P. de Roever, and R.T. Gerth. The μ-calculus as an assertion-language for fairness arguments. *Information and Computation*, 82:278–322, 1989.

[Sis91] A.P. Sistla. Proving correctness with respect to nondeterministic safety specifications. *Information Processing Letters*, 39(1):45–50, July 1991.

[Var87] M. Vardi. Verification of concurrent programs: The automata-theoretic framework. In *Proc. Symp. on Logic in Computer Science*. IEEE, 1987.

Appendix: Proofs

Proof of Theorem 1

Assume that P has a fair termination measure $\mu(p)$ and that $p_0, p_1, \cdots$ is an infinite computation. We must prove that $p_0, p_1, \cdots$ is unfair. To see this we let κ_i be the level of the active hypothesis of the transition $p_i \rightarrow p_{i+1}$ and we define $\kappa = \liminf_{i\to\infty} \kappa_i$, i.e. κ is the least value of κ_i that occurs infinitely often. Then from some point on κ_i is always at least κ, i.e. there is a K such that for all $i \geq K$, $\kappa_i \geq \kappa$.

It is not hard to see that $\kappa > 0$; in fact, if κ was 0, then the values of the T-measure would form a sequence $\mu^{\mathrm{T}}(p_0) \succeq \mu^{\mathrm{T}}(p_1) \succeq \cdots$ (by ($\mathrm{V_A}$) and ($\mathrm{V_{NoC}}$)),[4] where infinitely often the inequality is strict, namely each time $\kappa_i = 0$. This contradicts that $(W, \succ)$ is well-founded.

Thus $\kappa > 0$ and there is an ℓ such that for all $i \geq K$, the hypothesis at level κ is an ℓ-hypothesis (by ($\mathrm{V_{NoC}}$)) and this hypothesis is non-invalidating (by ($\mathrm{V_{NonI}}$)). It follows that ℓ is executed only finitely often. To see that the computation is unfair with respect to ℓ, we now only have to prove that ℓ is enabled infinitely often.

Assume the opposite is true. Thus for some $H \geq K$, it holds for all $i \geq H$ that ℓ is not enabled and

[4] $w \succeq w'$ means that $w = w'$ or $w \succ w'$.

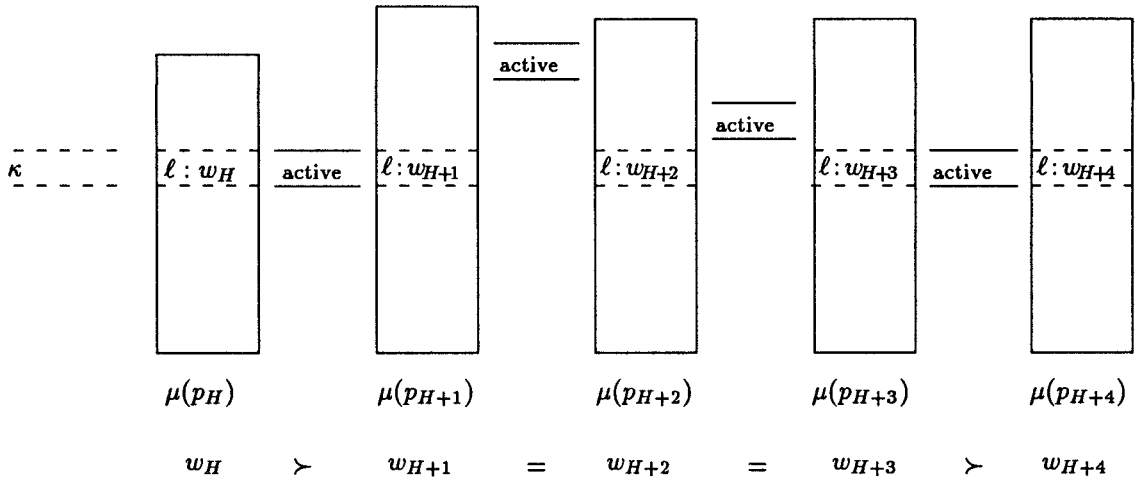

Figure 2: Soundness.

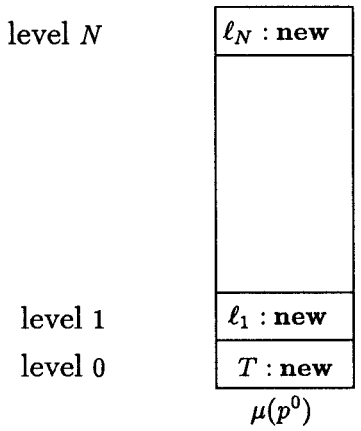

Figure 3: Initial stack.

the ℓ-hypothesis has the form $\ell : w_i$. As indicated in Figure 2, the values $w_i = \mu^\ell(p_i)$, $i \geq H$, give rise to an infinite descending sequence in W, because $w_H \succeq w_{H+1} \succeq \cdots$ (by (V_A) and (V_{NonI})) with strict inequalities whenever $\kappa_i = \kappa$. This contradicts that $(W, \succ)$ is well-founded.

Proof of Theorem 3

Assume that P fairly terminates and that there are N different commands. The proof is by a construction that defines the stack $\mu(p')$ in terms of the stack $\mu(p)$ when there is a transition $p \rightarrow p'$. Because the program is tree-like, this construction will define a unique value of μ for all p. The progress measures of the hypotheses take on values in a countable set W equipped with a relation $\succ$. Both W and $\succ$ are initially empty. The stack of p^0 is as illustrated in Figure 3. Here we created at levels 1 to N an hypothesis for each commands ℓ_i. The order of the hypotheses does not matter at this point. Each instance of **new** means that a new element is added to W. Hence creating the stack $\mu(p^0)$ results in there being $N+1$ elements in W, whereas $\succ$ remains empty.

When we create the stack $\mu(p)$ and use **new** to create a new element w at level κ, we define $\iota(w) = p$ and $\lambda(w) = \kappa$. Thus $\iota(w)$ denotes the program state where w is first used, and $\lambda(w)$ denotes the level where w is used.

Now assume that $\mu(p)$ has been defined and that there is a transition $p \rightarrow p'$ with ℓ denoting the command that is being executed. The idea behind the construction of $\mu(p')$ is to keep as much of $\mu(p)$ as possible. To state this more precisely we say that an ℓ'-hypothesis in $\mu(p)$ is *naturally active* if ℓ' is enabled in p or p' and the ℓ'-hypothesis is below the ℓ-hypothesis.

Case 1 If there is a naturally active hypothesis, let α be the naturally active hypothesis at the lowest level. The new stack becomes as illustrated in Figure 4. Here everything below α, indicated by S, is preserved. Also, the hypotheses above S are preserved, but their measures all change to new values.

Case 2 If there is no naturally active hypothesis, we let α be such that the α-hypothesis is the one just below the ℓ-hypothesis. Note that it may happen that $\alpha = \mathrm{T}$. The α-measure takes on a new value w', and we add $w \succ w'$ to the relation $\succ$ and say that α is *forced active*; in addition, the hypotheses above α are rotated one step downwards: Note that the ℓ is moved upwards (unless there is only one unfairness hypothesis in the stack) and that it is the only hypothesis moved upwards.

Whether $\mu(p')$ is constructed according to Case 1 or

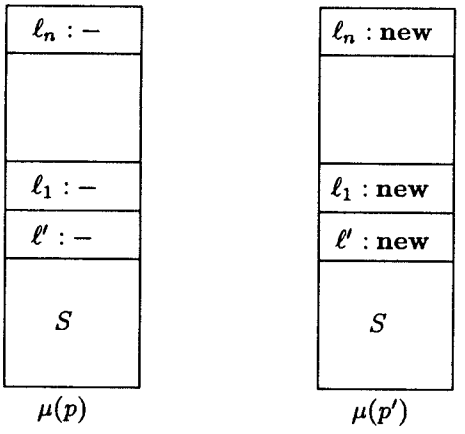

Figure 4: New stack in Case 1.

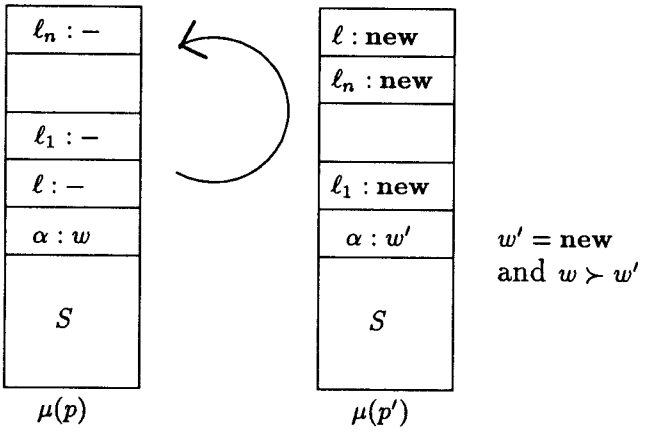

Figure 5: New stack in Case 2.

Case 2 above, the requirements (V_A), (V_{NonI}), and (V_{NoC}) can be seen to be satisfied for the transition $p \to p'$. Note also that when α is an active hypothesis, then there are no hypotheses below that can be active. Thus to finish the completeness proof we only need to show that $(W, \succ)$ is well-founded. We use the following properties:

Claim 1 If $p \to p'$, $\iota(w) \neq p'$, and $\mu^\alpha(p') = w$, then $\mu^\alpha(p) = w$ and the position of the α-hypothesis did not change on $p \to p'$. Moreover, if α is a label ℓ, then ℓ is not enabled and not executed on $p \to p'$. Also, the hypothesis just above the α-hypothesis in $\mu(p)$ does not change position and it is non-invalidated on $p \to p'$.

Proof By considering Case 1 and Case 2 above. □

Claim 2 Let w and w' be elements of W such that $w \succ w'$ and let $\kappa = \lambda(w)$. Let α by the hypothesis at level κ in $\mu(\iota(w))$.

(a) There is a path $\mathcal{P}^{w,w'} = p_0, \ldots p_n$ with $p_0 = \iota(w)$ and $p_n = \iota(w')$ such that the active level for $p_i \to p_{i+1}$ is greater than κ for $i < n-1$, and such that for $p_{n-1} \to p_n$ the hypothesis α is forced active.

(b) Moreover, no command of an hypothesis at or below κ is enabled along $\mathcal{P}^{w,w'}$.

Proof (a) By Claim 1, every p such that $\mu^\alpha(p) = w$ is reachable from $\iota(w)$ along a path where

- α is at level κ,
- if $\alpha = \ell \neq \mathrm{T}$, then ℓ is not executed and ℓ is not enabled, and
- $\mu^{\ell_\bullet}$ has the constant value w.

Since P is tree-like, there is a unique path $p_0, \ldots, p_{n-1}$ with $p_0 = \iota(w)$ such that there is a transition $p_{n-1} \to p_n = \iota(w')$, where $\mu(p_n)$ is constructed according to Case 2 and α is the hypothesis forced active.

(b) This follows from the choice of active hypothesis in Case 1 and Case 2. □

Now assume that there is an infinite descending sequence $w_0 \succ w_1 \succ \cdots$ in W. By (a) of the Claim, an infinite path $\mathcal{P}$ containing $\iota(w_0), \iota(w_1), \ldots$ can be put together from the paths $\mathcal{P}^{w_i,w_{i+1}}$. Along this path the active level is always at least $\kappa = \lambda(w_0) = \lambda(w_1) = \cdots$. Let α be the hypothesis at level κ. The commands that are executed infinitely often are above α, since (V_{NonI}) is satisfied. Also any command ℓ' that is executed only finitely often is eventually at level κ or below, because from the point where ℓ' is no longer executed, the ℓ'-hypothesis can only move downwards in the stack and will eventually settle at some level; this level is at most κ, because the hypotheses above κ are rotated infinitely often, namely each time α is forced active.

By the assumption that P fairly terminates, there is a command ℓ that is executed finitely often and enabled infinitely often. By the previous argument, the ℓ-hypothesis is at level κ or below. But the ℓ-hypothesis being infinitely often enabled then contradicts (b) of the Claim. Hence there are no infinite descending sequences in W, i.e. $(W, \succ)$ is well-founded. □

Proof of Theorem 2

The idea of the proof is similar to the use of the Sewing Lemma in [Kla92] for the immediate determinacy result of certain infinite games.

Assume that P fairly terminates. Also assume that there is a function $\mathcal{L}$ such that on any transition $p \rightarrow p'$, the value $\mathcal{L}(p')$ denotes the command executed in going from p to p' (the program state space and transition relation can always be extended to contain this information). By adding a history variable to P, we obtain a program $\overline{P}$, which also fairly terminates. A state of $\overline{P}$ is on the form $\sigma = \langle p_1, \ldots, p_n \rangle$ and the transitions of $\overline{P}$ are on the form $\langle p_1, \ldots, p_n \rangle \rightarrow \langle p_1, \ldots, p_{n+1} \rangle$, where $p_n \rightarrow p_{n+1}$ is a transition of P. The initial state of $\overline{P}$ is $\langle p^0 \rangle$, where p^0 is the initial state of P. For $\sigma = \langle p_1, \ldots, p_n \rangle$ define $p\sigma = p_n$. The set of states of $\overline{P}$ form a tree with root $\langle p^0 \rangle$. If $p \rightarrow p'$ and $p\sigma = p$, then the state $\sigma \cdot p'$ (which is list gotten by appending p' to the right end of σ) is a *child* of σ. A state σ is an *ancestor* of a state σ' if there are $\sigma_0, \ldots, \sigma_n$ such that σ_{i+1} is a child of σ_i for $i < n$ and $\sigma_0 = \sigma$ and $\sigma_n = \sigma'$. Define $\overline{\mathcal{L}}(\sigma) = \mathcal{L}(p\sigma)$. Let $\overline{\mu}$ designate the fairness measure given by the completeness proof of Theorem 3. The mapping $\overline{\mu}$ can be specified by a mapping α that to each σ associates a list $\overline{\alpha}(\sigma) = \langle \mathrm{T}, \ell_1, \ldots, \ell_N \rangle$ specifying the ordering of the hypotheses in the stack $\overline{\mu}(\sigma)$ and by a mapping $\overline{\theta} : \overline{P} \rightarrow W^N$ specifying for each σ a list $\mathbf{w} = \langle w_0, \ldots, w_N \rangle$ denoting the values of the progress measures at levels 0 to $N+1$. For a list $\mathbf{w}$, the i'th component is denoted $\mathbf{w}[i]$ and the sublist consisting of components from i to j is denoted $\mathbf{w}[i..j]$. We may assume that $(W, \succ)$ is totally ordered, i.e. is a well-ordering. We define an ordering, also denoted $\succ$, on W^{N+1} by $\mathbf{w} \succ \mathbf{w}'$ if for some i, $\mathbf{w}[i] > \mathbf{w}'[i]$, and for all $j < i$, $\mathbf{w}[j] = \mathbf{w}'[j]$. Then $\succ$ is a well-ordering. Now define $\theta(p) = \overline{\theta}(\sigma)$ and $\alpha(p) = \overline{\alpha}(\sigma)$, where σ is chosen such that $p\sigma = p$ and $\overline{\theta}(\sigma)$ is minimal with respect to $\succ$.

Claim 3 If $\overline{\theta}(\sigma)[0..n] = \overline{\theta}(\sigma')[0..n]$, then $\overline{\alpha}(\sigma)[0..n+1] = \overline{\alpha}(\sigma')[0..n+1]$.

Proof This follows from Claim 1. □

For $\mathbf{w}, \mathbf{w}' \in W^{N+1}$, define $|\mathbf{w}, \mathbf{w}'| = h$, where h is maximal such that for all $j \leq h$, $\mathbf{w}[j] = \mathbf{w}'[j]$.

Now consider a transition $p \rightarrow p'$ of P and let us prove that there is an α-hypothesis such that ($\mathrm{V_A}$), ($\mathrm{V_{NonI}}$), and ($\mathrm{V_{NoC}}$) are fulfilled. Let $\mathbf{w} = \theta(p)$ and $\mathbf{w}' = \theta(p')$. Then $\mathbf{w} = \overline{\theta}(\sigma)$ for some σ such that $p\sigma = p$. Let $\mathbf{w}'' = \overline{\theta}(\sigma \cdot p')$. By definition of $\theta(p')$, $\mathbf{w}'' \succeq \mathbf{w}'$. Also, let σ' be such that $\overline{\theta}(\sigma' \cdot p') = \theta(p')$.

There are two cases:

Case $\mathbf{w} \succ \mathbf{w}'$: Let $h = |\mathbf{w}, \mathbf{w}'|$. By Claim 3, $\alpha(p)[0..h+1] = \alpha(p')[0..h+1]$. Thus $\alpha(p)[h+1]$ is active and the stack below is unchanged, whence ($\mathrm{V_A}$) and ($\mathrm{V_{Noc}}$) are satisfied.

By definition of h, $\overline{\theta}(\sigma' \cdot p')[0..h] = \overline{\theta}(\sigma)[0..h]$. Thus the values in $\overline{\theta}(\sigma' p')[0..h]$ are created in an ancestor of $\sigma' \cdot p'$ and therefore $\overline{\theta}(\sigma')[0..h] = \overline{\theta}(\sigma' p')[0..h]$ by Claim 1. Also by Claim 1, it follows that $\mathcal{L}(p') = \overline{\mathcal{L}}(\sigma' \cdot p')$—the command executed on $p \rightarrow p'$—is not among $\overline{\alpha}(\sigma' \cdot p')[0..h+1] = \alpha(p')[0..h+1]$, whence ($\mathrm{V_{NonI}}$) is satisfied.

Case $\mathbf{w} \prec \mathbf{w}'$: We have $\mathbf{w}'' \succeq \mathbf{w}' \succeq \mathbf{w}$. Let $h = |w, w'|$. By construction of $\overline{\mu}$, the hypothesis at level $h+1$ is naturally active for the transition $\sigma \rightarrow \sigma \cdot p'$ of $\overline{P}$. But since $\mathbf{w}'' \succeq \mathbf{w}' \succeq \mathbf{w}$, it can be seen that $\mathbf{w}''[0..h] = \mathbf{w}'[0..h] = \mathbf{w}[0..h]$. It follows that $\alpha(p)[0..h+1] = \alpha(p')[0..h+1]$ and that the $\alpha(p)[h+1]$-hypothesis is naturally active for $p \rightarrow p'$ of P, whence ($\mathrm{V_A}$) and ($\mathrm{V_{NoC}}$) are satisfied. It can be seen in the same manner as in the previous case that ($\mathrm{V_{NonI}}$) is also satisfied □

Proof of Theorem 4

Given an effectively represented program P and a program state p, it is possible to calculate the sequence of program states, starting at the initial state, that leads to p. Thus the tree can be effectively traversed (even if it is infinitely branching). This traversal ensures that each time "**new**" is invoked, a unique progress value is returned. For example, we can represent W using the natural numbers; successive invocations of "**new**" then gives progress values '0,' '1,'...Note that the relation $\succ$ calculated on W is not the usual ordering on the natural numbers. Given a state p, the tree is traversed until p is encountered. At any program state, the value of the stack is calculated according to the the procedure given in the proof of Theorem 3. It follows that the value of the fair semi-measure at p can be recursively calculated. Similarly the relation $i \succ j$ can be seen to be recursive. By standard techniques of computability theory, the above procedure can be expressed formally as a recursive function h satisfying the properties in the statement of the Theorem. □