Fairness and Hyperfairness

Leslie Lamport

26 May 2000

Abstract

The notion of fairness in trace-based formalisms is examined. It is argued that, in general, fairness means machine closure. The notion of hyperfairness introduced by Attie, Francez, and Grumberg is generalized to arbitrary action systems. Also examined are the fairness criteria proposed by Apt, Francez, and Katz.

Contents

1 Introduction

Fairness in concurrent systems has been discussed for decades. There is even a book on the subject [8]. The best-known attempt to characterize fairness was probably Apt, Francez, and Katz's definition of three criteria for fairness notions [5]. We have learned much about trace-based formalisms since then, and I believe the definition of fairness is now fairly obvious. While any precise formalization of a vague concept is open to dispute, there does seem to be only one language-independent definition that distinguishes fairness from liveness. That definition appears to have been mentioned only by Abadi and Lamport [2, page 89] and Schneider [17, pages 254–257]. My purpose here is to re-examine the question of fairness in light of what has been learned in the last ten years.

Fairness for properties in an arbitrary trace-based formalism is defined in Section 2. Fairness usually appears in the guise of fairness conditions on actions in action systems. Section 3 reviews action systems—a simple abstraction that covers programs and many kinds of specifications—and recalls the definition of weak and strong fairness for actions. Section 4 generalizes weak and strong fairness and defines hyperfairness, a very strong form of fairness for actions. Hyperfairness generalizes the concept of the same name introduced by Attie, Francez, and Grumberg [6]. Section 5 extends the criteria of Apt, Francez, and Katz to arbitrary action systems and discusses these criteria. A concluding section muses upon what it all means.

Much of the material presented here is a review of well-known concepts; its presentation is quite terse. More loquacious expositions can be found in the cited literature. Proofs of the new results are not hard and are left mostly to the reader.

2 Fairness and Machine Closure

I assume a trace-based semantics in which a behavior is a sequence[1] of states and a property is a predicate on behaviors. Everything translates easily to a formalism (such as I/O automata [13]) in which a behavior has action names attached to each state transition.[2] In the informal discussion, I identify a

[1] Infinite sequences must be allowed; it doesn't matter if finite sequences are.

[2] An easy way to make the translation is to introduce a state variable whose value is the name of the transition just completed.

property with the set of behaviors satisfying it, so $R \Rightarrow T$ and $R \subseteq T$ are two ways of asserting that if a behavior satisfies property R, then it satisfies property T. However, formally, a property is a predicate on behaviors. Properties can be expressed as formulas in a temporal logic, but in this section, I make no assumption about how they are expressed.

The meaning of a system specification[3] is a property—namely, the set of behaviors representing a correct system execution. Any statement about specifications that is independent of the particular language in which the specification is written must be a statement about properties.

A *safety property* is one that is satisfied by an infinite behavior iff it is satisfied by each finite prefix of the behavior [4].[4] With the standard topology on sequences,[5] safety properties are closed sets. Let $\mathcal{C}(R)$ be the closure of property R in this topology, so $\mathcal{C}(R)$ is the strongest safety property implied by R.

A property L is a *liveness property* iff any finite prefix of a behavior can be extended to an infinite behavior that satisfies L. In the standard topology on sequences, liveness properties are dense sets. A theorem of topology implies that every property can be written as the conjunction of a safety property and a liveness property [4].

A pair $\langle S, L \rangle$ of properties is *machine closed* [1] iff S is equivalent to $\mathcal{C}(S \wedge L)$. This means that $\langle S, L \rangle$ is machine closed iff every finite sequence satisfying S can be extended to an infinite sequence satisfying $S \wedge L$.

Most methods for writing specifications, including CCS, Unity, and I/O automata, use some form of automaton to specify a safety property. The automaton asserts the property S consisting of all state sequences that the automaton can generate. In some of these methods, one also specifies a liveness property L—either implicitly (as in Unity), or by writing L explicitly (as in I/O automata). I will call the specification machine closed iff $\langle S, L \rangle$ is machine closed.

Machine closure of an automaton-based specification means that, as long as the automaton behaves correctly (keeps its safety property S satisfied), it

[3] I take the term *specification* to include any kind of precise description of a reactive system. For example, a program is a specification of what it means for the program to be executed correctly on a computer.

[4] In a formalism where behaviors are infinite sequences, a finite sequence satisfies a property R iff it is the prefix of some infinite sequence that satisfies R.

[5] This topology is defined by the distance function in which two sequences are a distance of $1/n$ apart if they are the same up to but not including their n^{th} states.

can never reach a state in which it is impossible to satisfy $S \wedge L$. In other words, the automaton can never "paint itself into a corner." A specification that purports to describe an implementation should be machine closed. However, one sometimes writes high-level specifications that are not machine closed. (An example is the specification of a serializable database in [10].)

In a number of methods, liveness is specified with so-called fairness conditions on the automaton. The common feature of all these conditions is that they produce machine-closed specifications. The only sensible definition of fairness that is independent of any specification language seems to be:

Definition 1 *A property L is a* fairness property for *property S iff $\langle S, L\rangle$ is machine closed.*

In this definition, the safety property[6] S plays the role of the automaton (or program) that appears, sometimes implicitly, in traditional notions of fairness.

We choose a fairness property L for a property S in order to write the specification $S \wedge L$. We can replace L by any other property $\widehat{L}$ such that $S \wedge \widehat{L}$ is equivalent to $S \wedge L$. (In this case, $\widehat{L}$ is a fairness property for S iff L is.) The strongest such $\widehat{L}$ is $S \wedge L$; the weakest is $\neg S \vee L$. It follows from simple results of topology that if $\langle S, L\rangle$ is machine closed, then $\neg S \vee L$ is a liveness property. Since, we can always replace a fairness property L with the liveness property $\neg S \vee L$, we could require that a fairness property be a liveness property—as most are in practise. However, it is simpler not to make this requirement.

Any property L is a fairness property for some safety property—in particular, for the safety property $\mathcal{C}(L)$. We are interested in whether L is a fairness property for a given safety property S. We don't care that there exists some other safety property for which L is a fairness property.

In the rest of this paper, we are concerned only with safety properties that are generated by action systems. Augmented with a method of hiding variables, action systems can express any safety property. However, we restrict our attention to fairness for action systems without hiding, which can express only a much more limited class of safety properties.

[6] The definition of machine closure implies that, if $\langle S, L\rangle$ is machine closed, then S is a safety property.

3 Fairness for Action Systems

An *action system*[7] consists of an initial state predicate *Init* and a set of predicates A_i on pairs of states. The A_i are called *system actions*. An action system expresses the safety property consisting of every behavior $\langle s_0, s_1, \ldots \rangle$ whose initial state s_0 satisfies *Init* and whose every pair $\langle s_n, s_{n+1} \rangle$ of successive states satisfies some system action.

I will describe action systems in terms of TLA (The Temporal Logic of Actions) [12]; it should be easy to translate the definitions and results into any other suitably expressive formalism. TLA assumes an underlying logic for writing state predicates and actions. An action is a predicate on pairs of states; it is written as a formula in which unprimed variables refer to the first state and primed variables refer to the second. Action A is defined to be a predicate on behaviors by letting $\langle s_0, s_1, \ldots \rangle$ satisfy A iff its initial step $\langle s_0, s_1 \rangle$ satisfies A. TLA includes the usual $\Box$ (forever) operator of linear-time temporal logic [14].

The safety property of an action system with initial predicate *Init* and system actions A_i is written in TLA as $\mathit{Init} \wedge \Box[\exists\, i : A_i]_v$, where v is the tuple of all variables occurring in *Init* or in the A_i.[8] For example, consider the action system with initial condition $x = y = 0$ and the two actions A_1, which increments x by 1, and A_2, which increments y by 1. These actions are defined formally by:

$$A_1 \;\triangleq\; (x' = x + 1) \wedge (y' = y) \qquad A_2 \;\triangleq\; (y' = y + 1) \wedge (x' = x) \qquad (1)$$

The safety property specified by this action system is:

$$(x = y = 0) \;\wedge\; \Box[A_1 \vee A_2]_{\langle x, y \rangle}$$

The subscript v (which equals $\langle x, y \rangle$ in the example) permits "stuttering" steps that do not change any relevant variables. This subscript is crucial for refinement and composition, but it is irrelevant when considering only a single specification. I will therefore omit all subscripts, and write $[A]$ instead of $[A]_v$, for an action A. The reader familiar with TLA should be able to figure out how to re-introduce the subscripts. However, it is important that

[7] The term "action system" was introduced in 1983 by Back and Kurki-Suonio [7], but the concept is much older.

[8] In TLA, an action system is expressed as a formula within the logic, rather than as a semantic restriction on the set of possible behaviors.

the safety property of an action system allow halting, which is represented by a behavior ending in an infinite sequence of stuttering steps. The action $[A]$ is true for any stuttering step.

For the rest of this section and for Section 4, let us assume a fixed action system with initial predicate *Init* and system actions A_i, and let us define N and S by

$$N \;\triangleq\; \exists\, i : A_i \qquad S \;\triangleq\; \mathit{Init} \wedge \Box[N] \tag{2}$$

Formula S is the system's safety property, and action N is called its *next-state relation.*

An action A is *enabled* in a state s iff there exists a state t such that $\langle s, t \rangle$ satisfies A. The operators WF and SF are defined by

$$\begin{aligned} \mathrm{WF}(A) &\;\triangleq\; \Diamond\Box(\textsc{Enabled}\, A) \Rightarrow \Box\Diamond A \\ \mathrm{SF}(A) &\;\triangleq\; \Box\Diamond(\textsc{Enabled}\, A) \Rightarrow \Box\Diamond A \end{aligned}$$

where ENABLED A is the predicate asserting that A is enabled. Formula WF(A), called *weak fairness* on A, asserts that if A eventually becomes enabled forever, then infinitely many A steps must occur. Formula SF(A), called *strong fairness* on A, asserts that if A is infinitely often enabled—even though it may also be infinitely often disabled—then infinitely many A steps must occur. Since eventually forever implies infinitely often, SF(A) implies WF(A) for any action A, so strong fairness is stronger than weak fairness.

Almost all fairness conditions for action systems that arise in real examples can be easily expressed as weak or strong fairness on actions. For example, the requirement

> If any of the actions $A_1, \ldots, A_k$ is ever enabled infinitely often,
> then a step satisfying one of those actions must eventually occur.

is just SF$(A_1 \vee \ldots \vee A_k)$. The following proposition, proved by Abadi and Lamport [3], shows that if A implies the next-state relation N, then weak and strong fairness on A are indeed fairness properties.

Proposition 1 *If S is defined by (2) and L is a finite or countably infinite conjunction of formulas of the form* WF(A) *and/or* SF(A)*, where each A implies N, then $\langle S, L \rangle$ is machine closed.*

4 Generalizing Weak and Strong Fairness

Operators GWF and GSF that generalize weak and strong fairness can be defined by replacing Enabled A with an arbitrary predicate P in the definitions of WF and SF:

$$\begin{array}{lcl} \text{GWF}(P, A) & \triangleq & \Diamond\Box P \Rightarrow \Box\Diamond A \\ \text{GSF}(P, A) & \triangleq & \Box\Diamond P \Rightarrow \Box\Diamond A \end{array}$$

(The concept of generalized fairness seems to have been defined first by Francez and Kozen [9].) Although these operators are not used in ordinary TLA specifications, they occur implicitly in TLA reasoning. When proving that a specification T_1 implies another specification T_2, we must substitute state functions for bound (hidden) variables of T_2. Let $\overline{F}$ denote the result of performing such a substitution on a formula F. Proving $T_1 \Rightarrow T_2$ requires proving that the fairness conditions of T_1 imply the barred fairness conditions of T_2, which may include formulas like $\overline{\text{WF}(B)}$. Since barring (which is just substitution) distributes over ordinary operators like $\Rightarrow$, $\Box$, and $\Diamond$, the formula $\overline{\text{WF}(B)}$ equals $\text{GWF}(\overline{\text{Enabled}\ B}, \overline{B})$. However, it need not equal $\text{WF}(\overline{B})$ because barring does not distribute over Enabled , so $\overline{\text{Enabled}\ B}$ does not necessarily equal Enabled $\overline{B}$ [12]. The same situation arises with SF formulas. Hence, we must prove that the WF and SF properties of T_1 imply the GWF and GSF properties of $\overline{T_2}$. The standard TLA rules for reasoning about WF and SF contain the appropriate barred formulas [12, Figure 5]. Those rules have straightforward generalizations in which all formulas of the form $\text{WF}(C)$ and $\text{SF}(C)$ are replaced by formulas $\text{GWF}(P, C)$ and $\text{GSF}(P, C)$, for arbitrary predicates P.

The properties $\text{GWF}(P, A)$ and $\text{GSF}(P, A)$ are not always fairness properties, even when A implies the next-state relation N. For example, $\text{GWF}(\text{true}, A)$ and $\text{GSF}(\text{true}, A)$ equal $\Box\Diamond A$. This property, called "unconditional fairness" on A, is not, in general, a fairness property. For example, $\langle S, \Box\Diamond\text{false}\rangle$ is machine closed iff S equals false.

I now give some necessary and sufficient conditions for GWF and GSF formulas to yield fairness properties. These conditions are not pretty, and expressing them requires some additional notation. Let "$\cdot$" be action composition, defined by letting $\langle s, t\rangle$ satisfy $A \cdot B$ iff there exists a state u such that $\langle s, u\rangle$ satisfies A and $\langle u, t\rangle$ satisfies B. Define $A^* \cdot B$ by

$$A^* \cdot B \;\triangleq\; B \,\vee\, (A \cdot B) \,\vee\, (A \cdot A \cdot B) \,\vee\, (A \cdot A \cdot A \cdot B) \,\vee\, \ldots$$

We now define three operators:[9]

$$\begin{array}{lcl}
h(N, A) & \triangleq & \textsc{Enabled}\,(N^* \cdot (N \wedge A)) \\
gw(P, N, A) & \triangleq & P \Rightarrow (h(N, A) \vee \textsc{Enabled}\,(N^* \cdot \neg P)) \\
gs(P, N, A) & \triangleq & P \Rightarrow (h(N, A) \vee \textsc{Enabled}\,(N^* \cdot \neg \textsc{Enabled}\,(N^* \cdot P)))
\end{array}$$

Since P implies $\textsc{Enabled}\,(N^* \cdot P)$, the monotonicity of $\textsc{Enabled}$ implies $gs(P, N, A) \Rightarrow gw(P, N, A)$.

The following proposition provides a necessary and sufficient condition for a GWF or GSF formula to be a fairness property.

Proposition 2 *If S is defined by (2) and L equals either* $\mathrm{GWF}(P, A)$ *or* $\mathrm{GSF}(P, A)$*, for state predicate P and action A, then $\langle S, L \rangle$ is machine closed iff S implies $\Box gw(P, N, A)$.*[10]

While not difficult, the proof of this proposition may help explain the rather obscure definitions of h and gw, so we sketch it here.

1. If $\langle S, \mathrm{GSF}(P, A) \rangle$ is machine closed, then so is $\langle S, \mathrm{GWF}(P, A) \rangle$.
   Proof: By the general result that $\langle S, F \rangle$ machine closed and $F \Rightarrow G$ imply $\langle S, G \rangle$ machine closed.
2. If $\langle S, \mathrm{GWF}(P, A) \rangle$ is machine closed, then $S \Rightarrow \Box gw(P, N, A)$.
   Proof: Assume S does not imply $\Box gw(P, N, A)$. Then there exists a finite behavior τ satisfying S whose last state τ_f satisfies $\neg h(N, A) \wedge \neg \textsc{Enabled}\,(N^* \cdot \neg P)$. Since τ_f satisfies $\neg h(N, A)$, by definition of h, this implies that every extension of τ satisfying $\Box N$ has no more A steps. Since τ_f satisfies $\neg \textsc{Enabled}\,(N^* \cdot \neg P)$, predicate P is forever true in every extension of τ satisfying $\Box N$. Hence every extension of τ satisfying S satisfies $\neg \mathrm{GWF}(P, A)$, contradicting the machine-closure assumption.
3. If $S \Rightarrow \Box gw(P, N, A)$, then $\langle S, \mathrm{GSF}(P, A) \rangle$ is machine closed.
   Proof: Let τ be a finite behavior satisfying S, with last state τ_f. The assumption means that τ_f satisfies either (i) $\neg P \vee \textsc{Enabled}\,(N^* \cdot \neg P)$ or (ii) $\textsc{Enabled}\,(N^* \cdot (N \wedge A))$. In case (i), we can extend τ to a behavior satisfying $S \wedge \mathrm{GSF}(P, A)$ by taking a finite (possibly null) sequence of N

[9] The "$P \Rightarrow$" in the definition of gw is redundant and is included for symmetry. All these operators can be expressed in terms of the weakest invariant operator win [11], since $\textsc{Enabled}\,(N^* \cdot B)$ is equivalent to $\neg win(N, \neg \textsc{Enabled}\, B)$, for any action B. A state predicate Q is considered to be an action by letting $\langle s, t \rangle$ satisfy Q iff s does.

[10] For a state predicate Q, the formula $S \Rightarrow \Box Q$ asserts that Q holds for every state of every behavior satisfying S.

steps, and then stuttering forever. In case (ii), we can extend τ with a finite sequence of N steps followed by an $N \wedge A$ step, and then repeat the construction, obtaining an infinite extension satisfying $S \wedge \mathrm{GSF}(P, A)$.

A specification usually requires the conjunction of fairness conditions for a set of actions, not just fairness for a single action. There seems to be no simple, weakest requirement for an arbitrary conjunction of GWF and GSF formulas to be a fairness property. However, the following is a rather powerful generalization of Proposition 1. Its proof is based on essentially the same construction used in step 3 in the proof above, except that τ must be repeatedly extended to satisfy the fairness properties for the different actions. This means that the stuttering construction can't be used, so we need the stronger *gs* formula for GSF properties. The detailed proof is similar to that of Proposition 1 and is omitted.

Proposition 3 *If S is defined by (2) and L is a finite or countably infinite conjunction of formulas, each of which is either (i) of the form* $\mathrm{GWF}(P, A)$ *where S implies $\Box gw(P, N, A)$ or (ii) of the form* $\mathrm{GSF}(P, A)$ *where S implies $\Box gs(P, N, A)$, then $\langle S, L\rangle$ is machine closed.*

It would seem appropriate to reserve the term hyperfairness for the strongest general fairness condition on an action that is a fairness property. This would mean finding, for an action A, the weakest predicate P for which $\mathrm{GSF}(P, A)$ is a fairness property. However, such a P does not, in general, exist. For example, define

$$\begin{array}{lcl@{\qquad\qquad}lcl} S & \triangleq & (x = 0) \wedge \Box(x' = x + 1) & Q(i) & \triangleq & x < i \\ A & \triangleq & x' = -7 & Q & \triangleq & \exists\, i \in Nat \, : \, Q(i) \end{array}$$

Any finite behavior satisfying S can be extended to one satisfying $S \wedge \Diamond\Box\neg Q(i)$, hence satisfying $S \wedge \mathrm{GSF}(Q(i), A)$, for any number i. Thus, $\mathrm{GSF}(Q(i), A)$ is a fairness property. Suppose there were a weakest P such that $\mathrm{GSF}(P, A)$ is a fairness property. Then each $Q(i)$ must imply P, so Q implies P. Hence, because GSF is antimonotonic in its first argument, $\mathrm{GSF}(P, A)$ implies $\mathrm{GSF}(Q, A)$, so $\mathrm{GSF}(Q, A)$ must be a fairness property. (See step 1 in the proof of Proposition 2.) But, S implies $\Box Q \wedge \Box\neg A$, which implies $\neg\mathrm{GSF}(Q, A)$, so $\mathrm{GSF}(Q, A)$ is not a fairness property. Hence, there can be no weakest P for which $\mathrm{GSF}(P, A)$ is a fairness property.

While there is no strongest fairness property $\mathrm{GSF}(P, A)$ for an arbitrary A, Proposition 2 and the definition of *gw* suggest taking P to be $h(N, A)$.

We therefore define the *hyperfairness* operator HF by

$$\text{HF}(N, A) \;\triangleq\; \text{GSF}(h(N, A), A)$$

While not in general the strongest possible fairness property for action A, it is still quite strong. It asserts that infinitely many A steps must occur if, infinitely often, a state is reached in which some possible sequence of N steps could enable A. Proposition 3 implies that the conjunction of any finite or countably infinite collection of hyperfairness properties is a fairness property.

The definitions of S and $h(N, A)$ imply that, for any behavior $\langle s_0, s_1, \ldots \rangle$ satisfying S and for any n, if s_n satisfies $h(N, A)$, then s_m satisfies $h(N, A)$ for every $m < n$. This implies:

Proposition 4 *If S is defined by (2), then S implies that $\Box\Diamond h(N, A)$ is equivalent to $\Box h(N, A)$, for any action A.*

This proposition shows that S implies the equivalence of GSF$(h(N, A), A)$ and GWF$(h(N, A), A)$, so it doesn't matter whether we use GSF or GWF in the definition of HF.

Attie, Francez, and Grumberg defined a tiny toy programming language called *IP*, based on multi-party CSP-style synchronization, and they defined hyperfairness for IP programs as follows [6]:

> **Definition** (Hyperfairness). If P is an *IP* program in which every top-level interaction is conspiracy-resistant, then an infinite computation π is *hyperfair* iff . . . . If P is an *IP* program in which not every top-level interaction is conspiracy-resistant, then every computation π of P is hyperfair.

The "..." is a condition that, in the context of the definition, is equivalent to the conjunction of hyperfairness properties for certain actions. It is in this sense that HF generalizes their definition of hyperfairness.

Hyperfairness asserts that something must happen infinitely often if it is infinitely often possible, where the "something" is a step that satisfies an action. Predicate reachability fairness, introduced by Queille and Sifakis [8, 16], makes the same assertion, except the "something" is the truth of a state predicate. Predicate reachability fairness is the special case of hyperfairness when the action is a state predicate (an action that depends only on the initial state).

5 The Criteria of Apt, Francez, and Katz

Apt, Francez, and Katz (henceforth called AFK) gave three "appropriateness" criteria for fairness notions in a programming language: feasibility, equivalence robustness, and liveness enhancement [5]. Although their abstract promised to consider "relations among various languages and models for distributed computation", they discussed mainly programs written in CSP-like languages with multi-party interactions. I now generalize their criteria to arbitrary action systems, remaining as faithful as possible to AFK's intentions and notation. Afterwards, I discuss the criteria and show that they are satisfied by a large class of hyperfairness properties.

5.1 The Criteria Redefined

For any action system $\mathbf{P}$, let $N_{\mathbf{P}}$ and $S_{\mathbf{P}}$ be the action N and property S defined by (2), and let $\mathbf{comp}(\mathbf{P})$ be the property $S_{\mathbf{P}} \wedge \mathrm{WF}(N_{\mathbf{P}})$. Assume some class $\mathcal{A}$ of action systems. A *fairness notion* $\mathbf{F}$ is a mapping that assigns a subset $\mathbf{F}(\mathbf{P})$ of $S_{\mathbf{P}}$ to every system $\mathbf{P}$ in $\mathcal{A}$ such that $\mathbf{F}(\mathbf{P})$ contains all finite computations in $S_{\mathbf{P}}$.

AFK defined the fairness notion $\mathbf{F}$ to be *feasible* iff $\langle S_{\mathbf{P}},\ \mathrm{WF}(N_{\mathbf{P}}) \wedge \mathbf{F}(\mathbf{P}) \rangle$ is machine closed, for every $\mathbf{P}$ in $\mathcal{A}$.

AFK defined their second criterion, equivalence robustness, only when all system actions are deterministic and mutually disjoint. In that case, a behavior $\langle s_0, s_1, \ldots \rangle$ in $S_{\mathbf{P}}$ is uniquely determined by the initial state s_0 and the sequence $\langle A_{k(0)},\ A_{k(1)}, \ldots \rangle$ of system actions such that $\langle s_n,\ s_{n+1} \rangle$ satisfies $A_{k(n)}$, for each n. We can therefore consider a behavior to consist of an initial state and a sequence of system actions. For behaviors π and ρ, AFK defined

> $\pi \equiv \rho$ iff π can be obtained from ρ by (possibly infinitely many)
> simultaneous transpositions of two independent [system] actions.

where system actions A_i and A_j are *independent* iff they commute—that is, iff $A_i \cdot A_j$ is equivalent to $A_j \cdot A_i$. We can then define a property R to be equivalence robust for the system $\mathbf{P}$ iff, for any pair of behaviors $\langle \pi, \rho \rangle$ such that $\pi \equiv \rho$, behavior π satisfies R iff behavior ρ does. AFK defined $\mathbf{F}$ to be equivalence robust iff $\mathbf{F}(\mathbf{P})$ is equivalence robust for $\mathbf{P}$, for all $\mathbf{P}$ in $\mathcal{A}$.

To extend this definition to arbitrary action systems, we need to generalize AFK's definition of $\equiv$. Without the assumption that system actions are pairwise disjoint and deterministic, it is not obvious what is meant by the simultaneous transpositions of infinitely many actions. The next paragraph defines $\pi \triangleright \rho$ to mean that π differs from ρ by the transposition of two adjacent actions, defines $\pi \rightarrow \rho$ to mean that ρ is obtained from π by a convergent sequence of such transpositions, and defines $\pi \equiv \rho$ to mean $\pi \rightarrow \rho$ and $\rho \rightarrow \pi$.

Let σ_i denote state i of a behavior σ, so σ equals $\langle \sigma_0, \sigma_1, \ldots \rangle$. Define $\pi \triangleright \rho$ to mean that there exists a natural number n such that, (i) $\pi_m = \rho_m$ for all $m \neq n+1$, and (ii) for any system actions A and B, if $\langle \rho_n, \rho_{n+1} \rangle$ satisfies A and $\langle \rho_{n+1}, \rho_{n+2} \rangle$ satisfies B, then $\langle \pi_n, \pi_{n+1} \rangle$ satisfies B and $\langle \pi_{n+1}, \pi_{n+2} \rangle$ satisfies A. This definition is illustrated pictorially below.[11]

$$\begin{tikzcd}
 & & \rho_{n+1} \arrow[dr, "B"] & \\
\cdots \;\; \pi_{n-1} = \rho_{n-1} \arrow[r] & \pi_n = \rho_n \arrow[ur, "A"] \arrow[dr, "B"'] & & \pi_{n+2} = \rho_{n+2} \;\; \cdots \\
 & & \pi_{n+1} \arrow[ur, "A"'] &
\end{tikzcd}$$

Let $\trianglerighteq$ mean $\triangleright$ or $=$. Define $\pi \rightarrow \rho$ to be true iff there exists an infinite sequence $\sigma^{(0)}, \sigma^{(1)}, \ldots$ of behaviors such that (i) $\sigma^{(0)} = \pi$, (ii) $\sigma^{(k)} \trianglerighteq \sigma^{(k+1)}$ for all k, and (iii) for every n there exists an m such that $\sigma_n^{(k)} = \rho_n$ for all $k \geq m$. Finally, let $\pi \equiv \rho$ equal $(\pi \rightarrow \rho) \wedge (\rho \rightarrow \pi)$

Having defined $\equiv$ for an arbitrary action system, we can define equivalence robustness as before: property R is equivalence robust for $\mathbf{P}$ iff $\pi \equiv \rho$ implies that π satisfies R iff ρ does; and $\mathbf{F}$ is equivalence robust iff $\mathbf{F}(\mathbf{P})$ is equivalence robust for $\mathbf{P}$, for all $\mathbf{P}$ in $\mathcal{A}$.

AFK's third criterion, liveness enhancement, essentially asserts that there is some $\mathbf{P}$ in $\mathcal{A}$ such that $\mathbf{comp}(\mathbf{P}) \wedge \mathbf{F}(\mathbf{P})$ is not equivalent to $\mathbf{comp}(\mathbf{P})$.[12]

5.2 The Criteria Re-examined

Feasibility of $\mathbf{F}$ is almost the same as requiring that $\mathbf{F}(\mathbf{P})$ be a fairness property for $S_{\mathbf{P}}$, for all $\mathbf{P}$ in $\mathcal{A}$. The two requirements are not the same because AFK made $\mathrm{WF}(N_{\mathbf{P}})$ an intrinsic assumption about an action system rather than just a particularly weak fairness property. This accords with the

[11] This condition implies that A and B commute for the pair $\langle \rho_n, \rho_{n+2} \rangle$ of states. We could further require that A and B simply commute—that is, commute for all pairs of states. It makes no difference to the ensuing discussion whether or not we change the definition of $\triangleright$, and hence of $\equiv$, in this way.

[12] AFK actually stated liveness enhancement in terms of terminating programs. As they observed, their definition is equivalent to this one for the particular class of programs they were considering.

common practice of calling "unfair" an execution of a multiprocess program that satisfies only this fairness property. While it is fruitless to argue with a definition, I believe that the fundamental nature of safety and machine closure suggests that it is unproductive to distinguish $\mathrm{WF}(N_{\mathbf{P}})$ from other fairness properties. Indeed, defining fairness to mean machine closure makes even TRUE a fairness property. *Ad hoc* restrictions to rule out such "trivial" fairness properties seem pointless.

Equivalence robustness of $\mathbf{F}$ requires that $\mathbf{F}(\mathbf{P})$ be equivalence robust for $\mathbf{P}$, for every $\mathbf{P}$ in $\mathcal{A}$. Unlike fairness, equivalence robustness depends on the actual action system $\mathbf{P}$, not just on its safety property $S_{\mathbf{P}}$. To show this dependence, I now construct two action systems $\mathbf{P}_1$ and $\mathbf{P}_2$ with equivalent safety properties (so the systems are semantically equivalent) and a property L that is equivalence robust for one and not the other.

Let $\mathbf{P}_1$ be the action system considered in Section 3 that has initial predicate $x = y = 0$ and actions A_1 and A_2 defined by (1); let $A_{2>}$ be the action $(x > y) \wedge A_2$; and let L be the property $\Box\Diamond A_{2>}$. Let π be the behavior obtained by alternately performing A_1 and A_2 actions, starting with an A_1 action; and let ρ be the same as π, except starting with an A_2 action. Since π can be obtained from ρ by interchanging each A_1 action with the following A_2 action, we have $\rho \equiv \pi$. Since $x > y$ holds in π at the beginning of each A_2 step and never holds in ρ, behavior π satisfies L and behavior ρ does not. Hence, L is not equivalence robust for $\mathbf{P}_1$.

Let $\mathbf{P}_2$ be the system with the same initial predicate $x = y = 0$ and the four actions $A_{1\leq}$, $A_{1>}$, $A_{2\leq}$, and $A_{2>}$, where the three new actions are defined by:

$$A_{1\leq} \;\triangleq\; (x \leq y) \wedge A_1 \qquad A_{1>} \;\triangleq\; (x > y) \wedge A_1 \qquad A_{2\leq} \;\triangleq\; (x \leq y) \wedge A_2$$

Property L, which asserts that infinitely many $A_{2>}$ actions occur, is obviously equivalence robust for $\mathbf{P}_2$.

Since $N_{\mathbf{P}_1}$ is equivalent to $N_{\mathbf{P}_2}$, the systems $\mathbf{P}_1$ and $\mathbf{P}_2$ have the same safety property. Thus, we have two semantically equivalent action systems and a property that is equivalence robust for one but not the other.

Under extremely weak hypotheses, we can show that for any action system $\mathbf{P}$, there exists a semantically equivalent action system (one defining the same safety property) $\widehat{\mathbf{P}}$ such that every property is equivalence robust for $\widehat{\mathbf{P}}$. We simply define $\widehat{\mathbf{P}}$ to have a separate system action for every pair of states that satisfies $N_{\mathbf{P}}$.

Since equivalence robustness is not a semantic property of a system, but depends on how the system is represented, it is unlikely to be a useful concept—except perhaps for action systems expressed in a language that severely restricts how they can be represented.

Liveness enhancement of $\mathbf{F}$, AFK's final criterion, asserts that there exists some $\mathbf{P}$ in $\mathcal{A}$ for which $\mathbf{F}(\mathbf{P})$ is stronger than $\mathrm{WF}(N_{\mathbf{P}})$. It rules out the fairness notion that assigns $\mathrm{WF}(N_{\mathbf{P}})$ to every system $\mathbf{P}$, reflecting AFK's decision not to consider $\mathrm{WF}(N_{\mathbf{P}})$ to be a fairness property. They may also have been trying to rule out trivial ways of defining a fairness notion that satisfies all three criteria. However, Attie, Francez, and Grumberg's definition of hyperfairness shows that there is a simple way to define $\mathbf{F}$ to satisfy all the criteria: (i) find a subclass of action systems for which there exists some equivalence-robust fairness property, and (ii) define $\mathbf{F}(\mathbf{P})$ to be that property if $\mathbf{P}$ is in the subclass, and to equal TRUE otherwise. For example, take the subclass consisting of those $\mathbf{P}$ for which $S_{\mathbf{P}}$ implies that every system action is always enabled, and define $\mathbf{F}(\mathbf{P})$ to equal $\forall\, i : \Box\Diamond A_i$ for all $\mathbf{P}$ in this subclass.

5.3 Hyperfairness and the AFK Conditions

Attie, Francez, and Grumberg proved that their definition of hyperfairness satisfies the three AFK conditions. Let's see if this is true of my definition of hyperfairness.

Feasibility follows directly from Proposition 3. To satisfy liveness enhancement, we would have to define a class of action systems and some particular conjunction of hyperfairness formulas for each action system in that class. This is a simple exercise and is omitted.

We are left with equivalence robustness. The conjunction of properties is equivalence robust if each conjunct is, so we need consider only individual hyperfairness formulas. For an arbitrary action A, the hyperfairness formula $h(N_{\mathbf{P}}, A)$ need not be equivalence robust for action system $\mathbf{P}$. For example, $H(N_{\mathbf{P}_1}, A_{2>})$ is not equivalence robust for $\mathbf{P}_1$, where $\mathbf{P}_1$ and $A_{2>}$ are the system and action defined above. However, we can prove the following result:

Proposition 5 *If* $\mathbf{P}$ *is an action system and* A *is the disjunction of system actions of* $\mathbf{P}$*, then* $\mathrm{HF}(N_{\mathbf{P}}, A)$ *is equivalence robust for* $\mathbf{P}$*.*

To prove the proposition, we assume $\pi \rightarrow \rho$ and π satisfies $\mathrm{HF}(N_{\mathbf{P}}, A)$, and we prove ρ satisfies $\mathrm{HF}(N_{\mathbf{P}}, A)$. We do this by proving (i) if π satisfies

$\Box\Diamond h(N_{\mathbf{P}}, A)$ then so does ρ and (ii) if ρ satisfies $\Box\Diamond A$ then so does π. Result (i) follows easily from Proposition 4; (ii) follows from the observation that, for any system action A_i and behaviors σ and τ with $\sigma \trianglerighteq \tau$, if infinitely many τ steps satisfy A_i, then infinitely many σ steps also satisfy A_i. The details are left as an exercise.

6 Conclusion

Fairness conditions are a way of expressing liveness properties, and liveness properties are inherently problematic. The question of whether a real system satisfies a liveness property is meaningless; it can be answered only by observing the system for an infinite length of time, and real systems don't run forever. Liveness is always an approximation to the property we really care about. We want a program to terminate within 100 years, but proving that it does would require the addition of distracting timing assumptions. So, we prove the weaker condition that the program eventually terminates. This doesn't prove that the program will terminate within our lifetimes, but it does demonstrate the absence of infinite loops.

In practice, almost all reactive systems can be specified using action systems together with simple weak and strong fairness properties.[13] Most specifications are machine closed. A machine-closed specification can always be written as an action system together with fairness properties only on disjunctions of system actions. However, in some cases, this requires a complicated representation of each system operation as the disjunction of infinitely many actions. For those cases, as well as for writing non-machine-closed specifications, we can use formulas of the form WF(A) or SF(A) when A is not simply the disjunction of system actions. The more general properties expressible with the GWF and GSF operators are rarely needed. A hyperfairness formula HF(N, A) is a particularly obscure example of such a property, since $h(N, A)$ will be impossible to compute in any practical situation, if hyperfairness differs from strong fairness. It therefore seems safe to predict that hyperfairness will be of at most theoretical interest.

[13] Formally, they can be specified only with action-system formalisms that have some way of hiding internal state. Without such hiding, it is impossible to specify even simple FIFO buffering [18]. This theoretical impossibility does not seem to be a practical obstacle [15].

Acknowledgments

Martín Abadi helped me formulate the definition of $\equiv$. He and Fred Schneider suggested improvements to the presentation. Nissim Francez pointed out to me the earlier definition of generalized fairness.

References

[1] Martín Abadi and Leslie Lamport. The existence of refinement mappings. *Theoretical Computer Science*, 82(2):253–284, May 1991.

[2] Martín Abadi and Leslie Lamport. Composing specifications. *ACM Transactions on Programming Languages and Systems*, 15(1):73–132, January 1993.

[3] Martín Abadi and Leslie Lamport. An old-fashioned recipe for real time. *ACM Transactions on Programming Languages and Systems*, 16(5):1543–1571, September 1994.

[4] Bowen Alpern and Fred B. Schneider. Defining liveness. *Information Processing Letters*, 21(4):181–185, October 1985.

[5] Krzysztof R. Apt, Nissim Francez, and Shmuel Katz. Appraising fairness in languages for distributed programming. *Distributed Computing*, 2:226–241, 1988.

[6] Paul C. Attie, Nissim Francez, and Orna Grumberg. Fairness and hyperfairness in multi-party interactions. *Distributed Computing*, 6(4):245–254, 1993.

[7] R. J. R. Back and R. Kurki-Suonio. Decentralization of process nets with centralized control. In *Proceedings of the SEcond Annual ACM Symposium on Principles of Distributed Computing*, pages 131–142. The Association for Computing Machinery, 1983.

[8] Nissim Francez. *Fairness.* Texts and Monographs in Computer Science. Springer-Verlag, New York, Berlin, Heidelberg, Tokyo, 1986.

[9] Nissim Francez and Dexter Kozen. Generalized fair termination. In *Proceedings of the Eleventh Annual ACM Symposium on Principles of Programming Languages*, pages 46–53, January 1984.

[10] Leslie Lamport. A simple approach to specifying concurrent systems. *Communications of the ACM*, 32(1):32–45, January 1989.

[11] Leslie Lamport. *win* and *sin*: Predicate transformers for concurrency. *ACM Transactions on Programming Languages and Systems*, 12(3):396–428, July 1990.

[12] Leslie Lamport. The temporal logic of actions. *ACM Transactions on Programming Languages and Systems*, 16(3):872–923, May 1994.

[13] Nancy Lynch and Mark Tuttle. Hierarchical correctness proofs for distributed algorithms. In *Proceedings of the Sixth Symposium on the Principles of Distributed Computing*, pages 137–151. ACM, August 1987.

[14] Zohar Manna and Amir Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*. Springer-Verlag, New York, 1991.

[15] Jayadev Misra. Specifying concurrent objects as communicating processes. *Science of Computer Programming*, 14(2–3):159–184, 1990.

[16] J. P. Queille and J. Sifakis. Fairness and related properties in transition systems—a temporal logic to deal with fairness. *Acta Informatica*, 19:195–210, 1983.

[17] Fred B. Schneider. *On Concurrent Programming*. Graduate Texts in Computer Science. Springer, 1997.

[18] A. P. Sistla, E. M. Clarke, N. Francez, and A. R. Meyer. Can message buffers be axiomatized in linear temporal logic? *Information and Control*, 63(1/2):88–112, October/November 1984.