FINITARY FAIRNESS

by

Rajeev Alur and Thomas A. Henzinger

Memorandum No. UCB/ERL M98/18

15 April 1998

COVER PAGE

FINITARY FAIRNESS

by

Rajeev Alur and Thomas A. Henzinger

Memorandum No. UCB/ERL M98/18

15 April 1998

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Finitary Fairness

RAJEEV ALUR
University of Pennsylvania and Bell Labs
and
THOMAS A. HENZINGER
University of California at Berkeley

Fairness is a mathematical abstraction: in a multiprogramming environment, fairness abstracts the details of admissible ("fair") schedulers; in a distributed environment, fairness abstracts the relative speeds of processors. We argue that the standard definition of fairness often is unnecessarily weak and can be replaced by the stronger, yet still abstract, notion of finitary fairness. While standard weak fairness requires that no enabled transition is postponed forever, finitary weak fairness requires that for every computation of a system there is an unknown bound k such that no enabled transition is postponed more than k consecutive times. In general, the *finitary restriction* $fin(F)$ of any given fairness requirement F is the union of all ω-regular safety properties contained in F.

The adequacy of the proposed abstraction is shown in two ways. Suppose we prove a program property under the assumption of finitary fairness. In a multiprogramming environment, the program then satisfies the property for all fair finite-state schedulers. In a distributed environment, the program then satisfies the property for all choices of lower and upper bounds on the speeds (or timings) of processors.

The benefits of finitary fairness are twofold. First, the proof rules for verifying liveness properties of concurrent programs are simplified: well-founded induction over the natural numbers is adequate to prove termination under finitary fairness. Second, the fundamental problem of consensus in a faulty asynchronous distributed environment can be solved assuming finitary fairness.

Categories and Subject Descriptors: D.1.3 [**Programming Techniques**]: Concurrent programming; D.2.4 [**Software Engineering**]: Software/program verification; F.3.1 [**Logics and Meanings of Programs**]: Specifying and verifying and reasoning about programs

General Terms: Formal verification, Distributed computing

Additional Key Words and Phrases: Modeling of asynchronous systems, Fairness, Program verification, Distributed consensus

Authors' addresses: Rajeev Alur, Department of Computer and Information Science, 200 South 33rd Street, University of Pennsylvania, Philadelphia, PA 19104. Email: alur@cis.upenn.edu. URL: `http://www.cis.upenn.edu/~alur`

Thomas A. Henzinger, Department of Electrical Engineering and Computer Sciences, 519 Cory Hall, University of California, Berkeley, CA 94720-1770. Email: tah@eecs.berkeley.edu. URL: `http://www.eecs.berkeley.edu/~tah`

A preliminary version of this paper appeared in the *Proceedings of the Ninth IEEE Symposium on Logic in Computer Science* (LICS), pp. 52–61, 1994. An extended version will appear in the *ACM Transactions on Programming Languages and Systems* (TOPLAS).

This research was supported in part by the ONR YIP award N00014-95-1-0520, by the NSF CAREER award CCR-9501708, by the NSF grant CCR-9504469, by the AFOSR contract F49620-93-1-0056, by the DARPA contract NAG2-892, and by the SRC contract 95-DC-324A.

1. INTRODUCTION

Formal methods for specification and verification offer a systematic approach to the design and implementation of concurrent systems, and have been demonstrated to be useful in detecting logical errors in early stages of design (see [Clarke and Wing 1996] for a survey). A key component of a formal approach is the mathematical modeling of concurrency. A popular abstraction for modeling concurrent computation is the interleaving assumption. In this approach, a computation of a concurrent system is obtained by letting, at each step, one of the enabled processes execute an atomic instruction. If all interleaving computations of a system satisfy a property, then the property holds for all implementations of the program independent of whether the tasks are multiprogrammed on the same processor and which scheduling policy is used, or whether the system is distributed and what the speeds of different processors are. Furthermore, the interleaving model is very simple as it reduces concurrency to nondeterminism.

The interleaving abstraction is adequate for proving safety properties of systems (a safety property is of the form "something bad never happens," for example, mutual exclusion). However, it is usually not suitable to prove guarantee properties (a guarantee property is of the form "something good will eventually happen," for example, termination) or more general liveness properties. The traditional approach to establishing guarantee properties is to require that all *fair* computations, instead of all computations, satisfy the property. Intuitively, fairness means that no individual process is ignored forever. Since all reasonable implementations of the system, whether in multiprogramming or in multiprocessing, are expected to be fair, if we prove that a program satisfies a property under the assumption of fairness, it follows that the property holds for all possible implementations of the program.

While the theory of specification and verification using different forms of fairness is well understood (see, for example, [Lehman et al. 1982; Francez 1986; Manna and Pnueli 1991]), fairness has two major drawbacks. First, the mathematical treatment of fairness, both in verification and in semantics, is complicated and requires higher ordinals. Second, fairness is too weak to yield a suitable model for fault-tolerant distributed computing. This is illustrated by the celebrated result of Fischer, Lynch, and Paterson that, under the standard fairness assumption, processes cannot reach agreement in an asynchronous distributed system if one process fails. We quote from their paper [Fischer et al. 1985]:

> These results do not show that such problems [distributed consensus] cannot be solved in practice; rather, they point out the need for more refined models of distributed computing that better reflect realistic assumptions about processor and communication timings.

We propose one such "more refined" model by introducing the notion of finitary fairness. We argue that finitary fairness (1) is sufficiently abstract to capture all possible implementations, both in the context of multiprogramming and in the context of distributed computing, and (2) does not suffer from either of the two aforementioned disadvantages associated with the standard notion of fairness.

Justification of finitary fairness

A fairness requirement is specified as a subset F of the set of all possible ways of scheduling different processes of a program. Let us first consider a multiprogramming environment, where all tasks are scheduled on a single processor. A scheduler that meets a given fairness requirement F is a program whose language (i.e., set of computations) is contained in F. The language of any program is a safety property. Furthermore, if the scheduler is finite-state, then its language is ω-regular. Thus, to capture all finite-state schedulers that implement F, it suffices to consider the (countable) union of all ω-regular safety properties that are contained in F. There are several popular definitions of F, such as weak fairness, strong fairness, etc. [Lehman et al. 1982; Francez 1986]. For every choice of F, we obtain its finitary version $fin(F)$ as the union of all ω-regular safety properties contained in F. In the case of weak fairness F, we show that the finitary version $fin(F)$ is particularly intuitive: while F prohibits a schedule if it postpones a task forever, $fin(F)$ also prohibits a schedule if there is no bound on how many consecutive times a task is postponed. In general, if the fairness requirement F is any ω-regular liveness property, we show that the finitary version $fin(F)$ is still live (but not necessarily ω-regular).

Now let us consider a distributed environment, where all tasks are executed concurrently on different processors. Here, finitary fairness corresponds to the assumption that the execution speeds of all processors stay within certain unknown, but fixed, bounds. Formally, a distributed system can be modeled as a transition system that imposes lower and upper time bounds on the transitions [Henzinger et al. 1994]. We show that a timed transition system satisfies a property for all choices of lower and upper time bounds iff the underlying untimed transition system satisfies the same property under finitary weak fairness. This correspondence theorem not only establishes the adequacy of finitary fairness for distributed systems, but in addition provides a method for proving properties of timed systems whose timing is not known a priori.

To summarize, finitary fairness abstracts the details of fair finite-state schedulers and the details of the independent speeds (timings) of processors with bounded drift. The parametric definition of finitary fairness also lends itself to generalizations such as *computable fairness*: the computable version $com(F)$ of a fairness requirement F is the (countable) union of all recursive safety properties that are contained in F. In a multiprogramming environment, computable fairness abstracts the details of fair computable schedulers; in a distributed environment, computable fairness abstracts the independent speeds of processors whose drift is bounded by any recursive function.

Benefits of finitary fairness

Program verification. We address the problem of verifying that a program satisfies a property under a finitary fairness requirement $fin(F)$. Since $fin(F)$ usually is not ω-regular, it may not be specifiable in temporal logic. This, however, is not an obstacle for verification. For finite-state programs, we show that a program satisfies a temporal-logic specification under $fin(F)$ iff it satisfies the specification under F itself. This means that for finite-state programs, the move to finitary fairness does not call for a change in the verification algorithm.

For general programs, the proof rules for verifying liveness properties are simplified if finitary fairness is used. Suppose we wish to prove that a program terminates. To prove that all computations of a program terminate, one typically identifies a ranking (variant) function from the states of the program to the natural numbers such that the rank decreases with every transition of the program. This method is not complete for proving the termination of all fair computations. First, there may not be a ranking function that decreases at every step. The standard complete verification rule, rather, relies on a ranking function that never increases and is guaranteed to decrease eventually [Lehman et al. 1982; Francez 1986]. For this purpose, one needs to identify so-called "helpful" transitions that cause the ranking function to decrease. Second, induction over the natural numbers is not complete for proving fair termination and one may have to resort to induction over ordinals higher than ω.

We show that proving the termination of a program under finitary weak fairness can be reduced to proving the termination of all computations of a transformed program. The transformed program uses a new integer variable, with unspecified initial value, to represent the bound on how many consecutive times an enabled transition may be postponed. Since the termination of all computations of the transformed program can be proved using a strictly decreasing ranking function on the natural numbers, reasoning with finitary fairness is conceptually simpler than reasoning with standard fairness.

Distributed consensus. A central problem in fault-tolerant distributed computing is the consensus problem, which requires that the nonfaulty processes of a distributed system agree on a common output value [Pease et al. 1980]. Although consensus cannot be reached in the asynchronous model if one process fails [Fischer et al. 1985], in practice, consensus is achieved in distributed applications using constructs like timeouts. This suggests that the asynchronous model with its standard fairness assumption is not a useful abstraction for studying fault-tolerance. One proposed solution to this problem considers the *unknown-delay model* (also called *partially synchronous model*) in which there is a fixed upper bound on the relative speeds of different components, but this bound is not known a priori [Dwork et al. 1988; Rhee and Welch 1992; Alur et al. 1997]. The asynchronous model with the finitary fairness assumption is an abstract formulation of the unknown-delay model. In particular, we prove that the asynchronous model with the finitary fairness assumption admits a wait-free solution for consensus that tolerates an arbitrary number of process failures, by showing that finitary fairness can replace the timing assumptions of the solution of [Alur et al. 1997].

The relationship between finitary fairness and the unknown-delay model is useful for two reasons. First, since the unknown-delay model is already popular in the distributed-computing community, it suggests that the proposed notion of finitary fairness is a reasonable one. Second, it implies that the proof rules for establishing temporal properties of programs under finitary fairness give, for the first time, a general methodology for reasoning about systems in the unknown-delay model.

Overview

The remainder of the paper is organized as follows. Section 2 gives an informal introduction to the definition of finitary fairness. Section 3 defines the operator *fin*, and presents its theoretical properties. In particular, it establishes connections between finitary fairness and the model of timed transition systems with unknown lower and upper bounds. Section 4 addresses verification under finitary fairness: model checking of finite-state programs, and proof rules for general programs. In particular, we show that proving termination of an arbitrary program under finitary fairness can be reduced to proving the termination of a transformed program without fairness assumptions. Section 5 illustrates the power of finitary fairness by presenting a wait-free solution to the distributed consensus problem. We conclude with Section 6, which summarizes the contributions.

2. INFORMAL MOTIVATION: BOUNDED FAIRNESS

Before introducing the general definition of finitary fairness (Section 3) and its applications (Sections 4 and 5), we begin by motivating the finitary version of weak fairness through the intuitive concept of bounded fairness. Consider the following simple program P_0 with a boolean variable x and an integer variable y:

$$\begin{array}{l}\textbf{initially } x = \mathit{true},\ y = 0;\\ \textbf{repeat } x := \neg x \textbf{ forever} \parallel \textbf{repeat } y := y + 1 \textbf{ forever}.\end{array}$$

The program P_0 consists of two processes, each with one transition. We use l and r to denote the two transitions: the transition l complements the value of the boolean variable x; the transition r increments the value of the integer variable y. A computation of P_0 is an infinite sequence of states, starting from the initial state ($x = \mathit{true}$ and $y = 0$), such that every state is obtained from its predecessor by applying one of the two transitions. For the purpose of this example, a *schedule* is an infinite word over the alphabet $\{l, r\}$. Each computation of P_0 corresponds, then, to a schedule, which specifies the order of the transitions that are taken during the computation. The two processes of P_0 can be executed either by multiprogramming or in a distributed environment.

Multiprogramming

In a multiprogramming environment, the two processes of P_0 are scheduled on a single processor. A *scheduler* is a set of possible schedules. One typically requires that the scheduler is "fair"; that is, it does not shut out one of the two processes forever. Formally, a schedule is *fair* iff it contains infinitely many l transitions and infinitely many r transitions; a scheduler is fair iff it contains only fair schedules.

Let $F_\infty \subseteq (l+r)^\omega$ be the set of fair schedules. If we restrict the set of computations of the program P_0 to those that correspond to fair schedules, then P_0 satisfies a property ϕ iff every computation of P_0 whose schedule is in F_∞ satisfies ϕ. For instance, under the fairness requirement F_∞, the program P_0 satisfies the property

$$\phi_\infty:\ \Box\Diamond\,(x = \mathit{true})\ \wedge\ \Box\Diamond\,\mathit{even}(y);$$

that is, in any fair computation, the value of x is true in infinitely many states, and the value of y is even in infinitely many states.[1] Note that there are computations of P_0 that correspond to unfair schedules, and do not satisfy the formula ϕ_∞. Thus, the fairness assumption is necessary to establish that the program P_0 satisfies the property ϕ_∞.

The fairness requirement F_∞ is an abstraction of all admissible real-life schedulers, namely, those that schedule each transition "eventually." Any (nonprobabilistic) real-life scheduler, however, is finite-state (i.e., uses finitely many variables, each of which can take only finitely many values) and therefore must put a bound on this eventuality. Consider, for instance, a round-robin scheduler that schedules the transitions l and r alternately. For round-robin schedulers, we can replace the fairness requirement F_∞ by the much stronger requirement F_1 that contains only two schedules, $(lr)^\omega$ and $(rl)^\omega$. Under F_1, the program P_0 satisfies the property

$$\phi_1 : \ \Box\Diamond\, (x = \mathit{true} \ \wedge \ \mathit{even}(y)),$$

which implies the property ϕ_∞. We call F_1 a 1-bounded scheduler. In general, for a positive integer k, a k-bounded scheduler never schedules one transition more than k times in a row. Formally, a schedule is ***k*-bounded**, for $k \geq 1$, iff it contains neither the subsequence l^{k+1} nor r^{k+1}: a scheduler is k-bounded iff it contains only k-bounded schedules (a similar definition is considered in [Jayasimha 1988]).

Let F_k be the set of k-bounded schedules. The assumption of k-boundedness is, of course, not sufficiently abstract, because for any k, it is easy to build a fair finite-state scheduler that is not k-bounded. So let us say a schedule is ***bounded*** iff it is k-bounded for some positive integer k, and a scheduler is bounded iff it contains only bounded schedules. Clearly, every fair finite-state scheduler is bounded. In order to prove a property of the program for all implementations, then, it suffices to prove the property for all bounded schedulers.

Let $F_\omega = \bigcup_{k\geq 1} F_k$ be the set of bounded schedules. If we restrict the set of computations of the program P_0 to those that correspond to bounded schedules, then P_0 satisfies a property ϕ iff every computation of P_0 whose schedule is in F_ω satisfies ϕ. We call F_ω the ***finitary restriction*** of the fairness requirement F_∞. Three observations about F_ω are immediate. First, the set F_ω is a proper subset of F_∞; in particular, the schedule $lrllrlllrllllr\ldots$ is fair but unbounded, and therefore belongs to $F_\infty \setminus F_\omega$. Second, the set F_ω itself is not a finite-state scheduler, but is the countable union of all fair finite-state schedulers. Third, F_ω is again a liveness property, in the sense that a stepwise scheduler cannot go wrong by making any finite number of scheduling decisions [Apt et al. 1988]: every finite word over $\{l, r\}$ can be extended into a bounded schedule.

Since the finitary fairness requirement F_ω is stronger than the fairness requirement F_∞, a program may satisfy more properties under F_ω. Consider, for example, the property

$$\phi_\omega : \ \Diamond\, (x = \mathit{true} \ \wedge \ \neg \mathit{power\text{-}of\text{-}2}(y)),$$

[1] The operators $\Box$ and $\Diamond$ are the standard temporal modalities of linear temporal logic. For a state predicate p, an infinite sequence of states satisfies the formula $\Box\, p$ ("always p") if all states satisfy p: the formula $\Diamond\, p$ ("eventually p") if some state satisfies p; and the formula $\Box\Diamond\, p$ ("always eventually p") if infinitely many states satisfy p. We refer the reader to [Manna and Pnueli 1991] for an introduction to temporal logic.

where the state predicate *power-of-2*(y) is true in a state iff the value of y is a power of 2. If a computation of P_0 does not satisfy ϕ_ω, then it must be the case that the transition l is scheduled only when *power-of-2*(y) holds. It follows that for every positive integer k, there is a subsequence of length greater than k that contains only r transitions. Such a schedule does not belong to F_ω and, hence, the program P_0 satisfies the property ϕ_ω under F_ω. On the other hand, it is easy to construct a fair schedule that does not satisfy ϕ_ω, which shows that P_0 does not satisfy ϕ_ω under F_∞.[2] ◂

Multiprocessing

In a distributed environment, the two processes of P_0 are executed simultaneously on two processors. While the speeds of the two processors may be different, one typically requires of a (nonfaulty) processor that each transition consumes only a finite amount of time. Again, the fairness requirement F_∞ is an abstraction of all admissible real-life processors, namely, those that complete each transition "eventually." Again, the fairness requirement F_∞ is unnecessarily weak.

Assume that the transition l, executed on Processor I, requires at least time ℓ_l and at most time u_l, for two unknown rational numbers ℓ_l and u_l with $u_l \geq \ell_l > 0$. Similarly, the transition r, executed on Processor II, requires at least time $\ell_r > 0$ and at most time $u_r \geq \ell_r$. Irrespective of the size of the four time bounds, there is an integer $k \geq 1$ such that both $k \cdot \ell_l > u_r$ and $k \cdot \ell_r > u_l$. Each computation corresponds, then, to a k-bounded schedule. It follows that finitary fairness is an adequate abstraction for speed-independent processors. It should be noted that finitary fairness is not adequate if the speeds of different processors can drift apart without a constant bound. For this case, we later generalize the notion of finitary fairness.

3. FORMAL PARAMETRIC DEFINITION: FINITARY FAIRNESS

3.1 Sets of infinite words

An *ω-language* over an alphabet Σ is a subset of the set Σ^ω of all infinite words over Σ. For instance, the set of computations of a program is an ω-language over the alphabet of program states. Note that the alphabet Σ need not be finite.

Regularity. An ω-language is *ω-regular* iff it is recognized by a Büchi automaton, which is a nondeterministic finite-state machine whose acceptance condition is modified suitably so as to accept infinite words [Büchi 1962]. The class of ω-regular languages is robust, with many alternative characterizations (see [Thomas 1990] for an overview of the theory of ω-regular languages). In particular, the set of models of any formula of (propositional) linear temporal logic (PTL) is an ω-regular language [Gabbay et al. 1980]. The set of computations of a finite-state program is an ω-regular language. The set F_∞ (Section 2) of fair schedules over the alphabet $\{l, r\}$ is an ω-regular language ($\Box\Diamond l \wedge \Box\Diamond r$), and so is the set F_k of k-bounded schedules, for every $k \geq 1$.

[2] It should also be noted that the set F_ω does not capture randomized schedulers. For, given a randomized scheduler that chooses at every step one of the two transitions with equal probability, the probability that the resulting schedule is in F_ω is 0. On the other hand, the probability that the resulting schedule is in F_∞ is 1.

Safety and liveness. For an ω-language $\Pi \subseteq \Sigma^\omega$, let $pref(\Pi) \subseteq \Sigma^*$ be the set of finite prefixes of words in Π. The ω-language Π is a ***safety property*** (or ***limit-closed***) iff for all infinite words $\overline{w}$, if all finite prefixes of $\overline{w}$ are in $pref(\Pi)$ then $\overline{w} \in \Pi$ [Alpern et al. 1986]. Every safety property Π is fully characterized by $pref(\Pi)$. Since a program can be executed step by step, the set of computations of a program is a safe ω-language over the alphabet of program states. A safety property is ω-regular iff it is recognized by a Büchi automaton all of whose states are accepting. For every $k \geq 1$, the set F_k of k-bounded schedules is an ω-regular safety property.

The ω-language Π is a ***liveness property*** iff $pref(\Pi) = \Sigma^*$ [Alpern and Schneider 1985]; that is, every finite word can be extended into a word in Π. The set F_∞ of fair schedules is an ω-regular liveness property.

Topological characterization. Consider the Cantor topology on infinite words: the distance between two distinct infinite words $\overline{w} = w_0 w_1 w_2 \ldots$ and $\overline{w}' = w'_0 w'_1 w'_2 \ldots$ is $1/2^i$, where i is the largest nonnegative integer such that $w_j = w'_j$ for all $0 \leq j < i$. The closed sets of the Cantor topology are the safety properties; the dense sets are the liveness properties. All ω-regular languages lie on the first two-and-a-half levels of the Borel hierarchy: every ω-regular language is in $\mathbf{F}_{\sigma\delta} \cap \mathbf{G}_{\delta\sigma}$.[3] There is also a temporal characterization of the first two-and-a-half levels of the Borel hierarchy [Manna and Pnueli 1990]. Let p be a past formula of PTL. Then every formula of the form $\Box p$ defines an $\mathbf{F}$-set; every formula of the form $\Diamond p$, a $\mathbf{G}$-set; every formula of the form $\Box\Diamond p$, a $\mathbf{G}_\delta$-set; and every formula of the form $\Diamond\Box p$, an $\mathbf{F}_\sigma$-set. For example, the set F_∞ of fair schedules is a $\mathbf{G}_\delta$-set.

3.2 The finitary restriction of an ω-language

Now we are ready to define the operator *fin*:

> The *finitary restriction* $fin(\Pi)$ of an ω-language Π is the (countable) union of all ω-regular safety properties that are contained in Π.

By definition, the finitary restriction of every ω-language is in $\mathbf{F}_\sigma$. Also, by definition, $fin(\Pi) \subseteq \Pi$. In the following, we list some additional properties of the operator *fin* in the form of propositions.

PROPOSITION 1. *Let Π and Π' be ω-languages. Then:*

(1) $fin(fin(\Pi)) = fin(\Pi)$.

(2) fin is monotonic: if $\Pi \subseteq \Pi'$, *then* $fin(\Pi) \subseteq fin(\Pi')$.

(3) fin distributes over intersection: $fin(\Pi \cap \Pi') = fin(\Pi) \cap fin(\Pi')$.

PROOF. The first two assertions follow immediately from the definition of *fin*. Since $\Pi \cap \Pi'$ is contained in Π as well as in Π', from the monotonicity property, we have $fin(\Pi\cap\Pi') \subseteq fin(\Pi) \cap fin(\Pi')$. To prove the inclusion $fin(\Pi)\cap fin(\Pi') \subseteq fin(\Pi\cap\Pi')$, consider a word $\overline{w} \in fin(\Pi)\cap fin(\Pi')$. From the definition of *fin*, there exist ω-regular safety properties $\Pi_1 \subseteq \Pi$ and $\Pi'_1 \subseteq \Pi'$ such that $\overline{w} \in \Pi_1$ and $\overline{w} \in \Pi'_1$. The class

[3] The first level of the Borel hierarchy consists of the class $\mathbf{F}$ of closed sets and the class $\mathbf{G}$ of open sets; the second level, of the class $\mathbf{G}_\delta$ of countable intersections of open sets and the class $\mathbf{F}_\sigma$ of countable unions of closed sets; the third level, of the class $\mathbf{F}_{\sigma\delta}$ of countable intersections of $\mathbf{F}_\sigma$-sets and the class $\mathbf{G}_{\delta\sigma}$ of countable unions of $\mathbf{G}_\delta$-sets.

of safety properties is closed under intersection, and so is the class of ω-regular languages. Hence, $\Pi_1 \cap \Pi'_1$ is an ω-regular safety property. Since $\overline{w} \in \Pi_1 \cap \Pi'_1$ and $\Pi_1 \cap \Pi'_1 \subseteq \Pi \cap \Pi'$, we have $\overline{w} \in fin(\Pi \cap \Pi')$, and (3) follows. □

The next proposition formalizes the claims we made about the example in Section 2. It also shows that the finitary restriction of an ω-regular language is not necessarily ω-regular.

PROPOSITION 2. *Let F_∞ be the set of fair schedules from Section 2, and let F_ω be the set of bounded schedules. Then F_ω is the finitary restriction of F_∞ (that is, $F_\omega = fin(F_\infty)$), and F_ω is neither ω-regular nor safe.*

PROOF. Recall that $F_\omega = \bigcup_{k \geq 1} F_k$, and $fin(F_\infty)$ is the union of ω-regular safety properties contained in F_∞. Each F_k is an ω-regular safety property and $F_k \subset F_\infty$. Hence, $F_\omega \subseteq fin(F_\infty)$.

Now consider an ω-regular safety property G contained in F_∞. Suppose that G is accepted by a Büchi automaton M_G over the alphabet $\{l, r\}$. Since G is a safety property, all states of M_G are accepting. It suffices to prove that if M_G has k states, then $G \subseteq F_k$. Suppose not. Then there is a word $\overline{w}$ such that M_G accepts $\overline{w}$ and $\overline{w}$ contains $k+1$ consecutive symbols of the same type, say l; that is, $\overline{w} = \overline{w}_1 l^{k+1} \overline{w}_2$ for $\overline{w}_1 \in (l+r)^*$ and $\overline{w}_2 \in (l+r)^\omega$. Since M_G has only k states, it follows that there is a state s of M_G such that there is a path from the initial state to s labeled with $\overline{w}_1 l^i$ for some $0 \leq i \leq k$, and there is a cycle that contains s and all of whose edges are labeled with l. This implies that M_G accepts also the word $\overline{w}_1 l^\omega$, which is not a fair schedule, a contradiction to the inclusion $G \subseteq F_\infty$.

Observe that, for each $k \geq 1$, the schedule l^k can be extended to a word in F_k, and hence $l^k \in pref(F_\omega)$. However, $l^\omega \notin F_\omega$, implying that F_ω is not limit-closed. Thus, F_ω is not a safety property.

Now we prove that F_ω is not ω-regular. Suppose that F_ω is ω-regular. From the closure properties of ω-regular languages, the set $G = F_\infty \setminus F_\omega$ of unbounded fair schedules is also ω-regular. We know that G is nonempty (it contains the schedule *lrllrlllrllllr* ...). From properties of the ω-regular languages it follows that G contains a word $\overline{w}$ such that $\overline{w} = \overline{w}_1 \overline{w}_2^\omega$ for two finite words $\overline{w}_1, \overline{w}_2 \in (l+r)^*$. Since $\overline{w} \in F_\infty$, the word $\overline{w}_2$ contains at least one l and one r symbol. This means that, for $k = |\overline{w}_1| + |\overline{w}_2|$, the word $\overline{w}$ is k-bounded, a contradiction to the assumption that $\overline{w} \notin F_\omega$. □

In other words, although F_ω is a countable union of safety properties that are definable in PTL, F_ω itself is neither a safety property nor definable in PTL. To define F_ω in temporal logic, one would need infinitary disjunction. However, F_ω is a liveness property, and thus suitable as a fairness assumption. In general, when applied to ω-regular properties, the operator fin preserves liveness. This is the contents of the next proposition.

PROPOSITION 3. *If Π is an ω-regular language, then $pref(fin(\Pi)) = pref(\Pi)$.*

PROOF. Since $fin(\Pi) \subseteq \Pi$ and $pref$ is monotonic, $pref(fin(\Pi)) \subseteq pref(\Pi)$. To prove the inclusion $pref(\Pi) \subseteq pref(fin(\Pi))$, suppose that Π is an ω-regular language over Σ, and consider $\overline{w} \in pref(\Pi)$. From the ω-regularity of Π, it follows that there is a word $\overline{w}' \in \Pi$ such that $\overline{w}' = \overline{w}\ \overline{w}_1 \overline{w}_2^\omega$ for finite words $\overline{w}_1, \overline{w}_2 \in \Sigma^*$.

The language containing the single word $\overline{w}'$ is ω-regular, safe, and contained in Π. Hence $\overline{w}' \in \mathit{fin}(\Pi)$, and therefore $\overline{w} \in \mathit{pref}(\mathit{fin}(\Pi))$. □

This immediately leads to the following corollary.

COROLLARY 4. *If Π is an ω-regular liveness property, then $\mathit{fin}(\Pi)$ is live.*

Observe that the language F_∞ is ω-regular and live, and hence, F_ω is also live: $\mathit{pref}(F_\omega) = (l + r)^*$. This means that after any finite number of steps of executing the programs, there is a way to produce an infinite execution that satisfies the fairness requirement F_ω, just as in case of the original requirement F_∞. While we have seen that the operator *fin* preserves safety (trivially) and liveness (Corollary 4) for ω-regular languages, this is not the case in general.

PROPOSITION 5.

(*1*) *There is an ω-regular language Π so that $\mathit{fin}(\Pi)$ is not ω-regular.*

(*2*) *There is a safety property Π so that $\mathit{fin}(\Pi)$ is not safe.*

(*3*) *There is a liveness property Π so that $\mathit{fin}(\Pi)$ is not live.*

PROOF. The first assertion follows immediately from Proposition 2.

For the second assertion, consider the infinite word $\overline{w} = lrllrlllrllllr\ldots$ and the ω-language $\Pi' = \{\overline{w}'r^\omega \mid \overline{w}'$ is a finite prefix of $\overline{w}\}$. The ω-language Π' is not limit-closed; its limit closure is the safety property $\Pi = \Pi' \cup \{\overline{w}\}$. We claim that $\mathit{fin}(\Pi) = \Pi'$. To see that $\Pi' \subseteq \mathit{fin}(\Pi)$, observe that for every infinite word $\overline{w}'' \in \Pi'$, the singleton set $\{\overline{w}''\}$ is an ω-regular safety property. To see that $\overline{w} \notin \mathit{fin}(\Pi)$, suppose that $\overline{w} \in G$ for some ω-regular safety property $G \subseteq \Pi$. But any Büchi automaton accepting G must also accept an infinite word of the form $\overline{w}_1\overline{w}_2^\omega$, where $\overline{w}_2$ contains an l. This word is not in Π, a contradiction to $G \subseteq \Pi$.

For the third assertion, consider the ω-language Π of all words over the alphabet $\{l, r\}$ that cannot be written in the form $\overline{w}_1\overline{w}_2^\omega$, for a finite word $\overline{w}_1 \in (l + r)^*$ and an infinite word $\overline{w}_2 \in (l + r)^\omega$. The ω-language Π is a liveness property, because $\overline{w}'\overline{w} \in \Pi$ for every finite word $\overline{w}' \in (l+r)^*$ and the infinite word $\overline{w} = lrllrlllrllllr\ldots$ We claim that $\mathit{fin}(\Pi) = \emptyset$, which is not a liveness property. Suppose that $\overline{w}'' \in \mathit{fin}(\Pi)$ for some infinite word $\overline{w}''$; that is, $\overline{w}'' \in G$ for some ω-regular safety property $G \subseteq \Pi$. Then the Büchi automaton that accepts G also accepts some infinite word of the form $\overline{w}_1\overline{w}_2^\omega$, a contradiction to $G \subseteq \Pi$. □

The operator *fin* is illustrated on some important ω-regular languages below:

$\mathit{fin}(\Diamond p) = \Diamond p$;
$\mathit{fin}(\Box p) = \Box p$;
$\mathit{fin}(\Diamond\Box p) = \Diamond\Box p$;
$\mathit{fin}(\Box\Diamond p) = \{\overline{w} \mid \exists k$: every subsequence of length k contains a $p\}$;
$\mathit{fin}(\Box\Diamond p \text{ — } \Box\Diamond q) = \{\overline{w} \mid \exists k$: every subsequence with k p's contains a $q\}$.

3.3 Transition systems: From standard fairness to finitary fairness

Concurrent programs, including shared-memory and message-passing programs, can be modeled as transition systems [Manna and Pnueli 1991]. A *transition system* P is a triple (Q, T, Q_0), where Q is a set of states, T is a finite set of transitions, and $Q_0 \subseteq Q$ is a set of initial states. Each state $q \in Q$ is an assignment of values to all program variables; each transition $\tau \in T$ is a binary relation on the states (that is, $\tau \subseteq Q^2$). For a state q and a transition τ, let $\tau(q) = \{q' \mid (q, q') \in \tau\}$ be the set of τ-successors of q. A *computation* $\overline{q}$ of the transition system P is an infinite sequence of states such that $q_0 \in Q_0$ and for every position $i \geq 0$, there is a transition $\tau \in T$ with $q_{i+1} \in \tau(q_i)$. We write $\Pi(P)$ for the set of computations of P.[4] The set $\Pi(P)$ is a safe ω-language over Q. If Q is finite, then $\Pi(P)$ is ω-regular.

A transition τ is *enabled* at the i-th position of a computation $\overline{q}$ iff $\tau(q_i)$ is nonempty, and τ is *taken* at the i-th position of $\overline{q}$ iff $q_{i+1} \in \tau(q_i)$. Without loss of generality, we assume that the set of program variables contains for every transition $\tau \in T$ a boolean variable $enabled(\tau)$ and a boolean variable $taken(\tau)$, whose values indicate, at each position of every computation, which transitions are enabled and which are taken. Let the *scheduling alphabet* Σ_T be the (finite) set of interpretations for these boolean variables; that is, Σ_T is the power set of the set $\{enabled(\tau), taken(\tau) \mid \tau \in T\}$. Given a computation $\overline{q}$ of P, the *schedule* $\sigma(\overline{q})$ of $\overline{q}$ is the projection of $\overline{q}$ to the scheduling alphabet. The set of schedules of P, then, is a safety property over Σ_T.

A *fairness requirement* F for the transition system P is an ω-language over the finite scheduling alphabet Σ_T. The fairness requirement restricts the set of legal computations of the program. Usually, F is an ω-regular liveness property [Francez 1986; Manna and Pnueli 1991]. The requirement of liveness ensures that, when executing a program, a fairness requirement does not become infeasible after any finite number of scheduling decisions [Apt et al. 1988].

In particular, the requirement of *weak fairness* WF for P is the set of all infinite words $\overline{w} \in \Sigma_T^\omega$ such that for every transition $\tau \in T$, there are infinitely many positions $i \geq 0$ with $taken(\tau) \in w_i$ or $enabled(\tau) \notin w_i$; that is, no transition is enabled forever without being taken. It is specified by the temporal-logic formula

$$\bigwedge_{\tau \in T} (\Diamond\Box\, enabled(\tau) \rightarrow \Box\Diamond\, taken(\tau)).$$

The requirement WF is ω-regular and live.

The requirement of *strong fairness* SF for P is the set of all infinite words $\overline{w} \in \Sigma_T^\omega$ such that for every transition $\tau \in T$, if there are infinitely many positions $i \geq 0$ with $enabled(\tau) \in w_i$, then there are infinitely many positions $j \geq 0$ with $taken(\tau) \in w_j$; that is, no transition is enabled infinitely often without being taken. It is a stronger requirement than weak fairness ($SF \subset WF$), and is specified by the formula

$$\bigwedge_{\tau \in T} (\Box\Diamond\, enabled(\tau) \rightarrow \Box\Diamond\, taken(\tau)).$$

The requirement SF is again ω-regular and live. However, while the weak-fairness requirement WF is a $\mathbf{G}_\delta$-set; the strong-fairness requirement SF is neither in $\mathbf{G}_\delta$ nor in $\mathbf{F}_\sigma$, but lies in $\mathbf{F}_{\sigma\delta} \cap \mathbf{G}_{\delta\sigma}$. Since both WF and SF are ω-regular liveness properties, their finitary restrictions $fin(WF)$ and $fin(SF)$, which belong to $\mathbf{F}_\sigma$, are again live (Corollary 4).

[4] Note that we consider only infinite, or nonterminating, executions to simplify the presentation. A terminating execution can be modeled as an infinite execution by letting the program execute a dummy (stutter) transition infinitely many times in the terminal state.

The next theorem (a generalization of Proposition 2) shows that the finitary restriction of weak and strong fairness coincide with the appropriate notions of bounded fairness. Define a schedule $\overline{w} \in \Sigma_T^\omega$ to be *weakly k-bounded*, for a positive integer k, iff for all transitions τ of P, the transition τ cannot be enabled for more than k consecutive positions without being taken; that is, for all positions $i \geq 0$, there is a position j, with $i \leq j \leq i+k$, such that either $taken(\tau) \in w_j$ or $enabled(\tau) \notin w_j$. A schedule is *weakly bounded* iff it is weakly k-bounded for some $k \geq 1$. Similarly, a schedule $\overline{w}$ is *strongly k-bounded* iff for all transitions τ, if a subsequence of $\overline{w}$ contains k distinct positions where τ is enabled, then it contains a position where τ is taken; that is, for all positions $0 \leq i_1 < i_2 < \cdots < i_k$, if $enabled(\tau) \in w_{i_j}$ for all $1 \leq j \leq k$, then $taken(\tau) \in w_i$ for some position i with $i_1 \leq i \leq i_k$. A schedule is *strongly bounded* iff it is strongly k-bounded for some $k \geq 1$.

THEOREM 6. *Consider a transition system with the transition set T and the weak and strong fairness requirements WF and SF. For all infinite words $\overline{w}$ over Σ_T, $\overline{w} \in fin(WF)$ iff $\overline{w}$ is weakly bounded, and $\overline{w} \in fin(SF)$ iff $\overline{w}$ is strongly bounded.*

PROOF. We will consider only the case of weak fairness. The set of weakly k-bounded schedules, for a fixed k, is defined by the formula

$$\bigwedge_{\tau \in T} \Box(\, wf(\tau) \vee \bigcirc wf(\tau) \vee \bigcirc^2 wf(\tau) \vee \cdots \vee \bigcirc^k wf(\tau)\,),$$

where $wf(\tau)$ stands for the disjunction $taken(\tau) \vee \neg enabled(\tau)$. It follows that the set of weakly k-bounded schedules is safe and ω-regular, for all $k \geq 0$. Thus, every weakly bounded schedule is in $fin(WF)$.

Now consider an ω-regular safety property G contained in WF. Suppose that G is accepted by a Büchi automaton M_G over the alphabet Σ_T. Since G is a safety property, all states of M_G are accepting. It suffices to prove that if M_G has k states, then every schedule in G is weakly k-bounded. Suppose not. Let us say that a symbol of Σ_T is weakly unfair to a transition τ if it contains $enabled(\tau)$ but does not contain $taken(\tau)$. By assumption, there is a word $\overline{w}$ accepted by M_G and a transition τ such that $\overline{w}$ contains $k+1$ consecutive symbols all of which are weakly unfair to τ. Since M_G has only k states, it follows that there is a cycle in M_G all of whose edges are labeled with symbols that are weakly unfair to τ. This implies that M_G accepts a schedule that is not weakly fair to τ, a contradiction to the inclusion $G \subseteq WF$. □

Observe that for the example program P_0 of Section 2, we have $T = \{l, r\}$, the propositions $enabled(l)$ and $enabled(r)$ are true in every state, and the fairness requirement F_∞ equals both WF and SF. This implies that there is a transition system with two transitions such that both $fin(WF)$ and $fin(SF)$ are neither ω-regular nor safe.

A computation $\overline{q}$ of the transition system P is *fair* with respect to the fairness requirement F iff $\sigma(\overline{q}) \in F$. We write $\Pi_F(P)$ for the set of fair computations of P. A *specification* Φ for the transition system P is a set of infinite words over the state alphabet Q. The transition system P *satisfies* the specification Φ under the fairness requirement F iff $\Pi_F(P) \subseteq \Phi$. If we prove that P satisfies Φ under the fairness requirement F, then P satisfies Φ for all implementations of F; if we prove that P satisfies Φ under the finitary restriction *fin*(F), then P satisfies Φ for all finite-state implementations of F. In Section 4, we show that proving the latter is conceptually simpler than proving the former.

3.4 Timed transition systems: From timing to finitary fairness

Standard models for real-time systems place lower and upper time bounds on the duration of delays [Merritt et al. 1991; Henzinger et al. 1994]. Since the exact values of the time bounds are often not known a priori, it is desirable to design programs that work for all possible choices of time bounds. It has long been realized that the timing-based model with unknown delays is different from, and often more appropriate than, the asynchronous model (with standard fairness) [Dwork et al. 1988; Rhee and Welch 1992; Alur et al. 1997]. We show that the unknown-delay model is equivalent to the asynchronous model with finitary fairness.

Real-time programs can be modeled as timed transition systems [Henzinger et al. 1994]. A *timed transition system* $P_{\ell,u}$ consists of a transition system $P = (Q, T, Q_0)$ and two functions ℓ and u from the set T of transitions to the set $\mathbb{Q}_{>0}$ of positive rational numbers. The function ℓ associates with each transition τ a lower bound $\ell_\tau > 0$; the function u associates with τ an upper bound $u_\tau \geq \ell_\tau$. The interleaving semantics of transition systems is extended to timed transition systems by labeling every state of a computation with a real-valued time stamp. A *time sequence* $\bar{t}$ is an infinite nondecreasing and unbounded sequence of nonnegative real numbers. For $\bar{t}$ to be consistent with a given computation $\bar{q}$ of the underlying transition system P, we require that a transition τ has to be enabled continuously at least for time ℓ_τ before it is taken, and it must not stay enabled continuously longer than time u_τ without being taken. We assume that the transition from state q_i to state q_{i+1}, for all $i \geq 0$, is taken at time t_{i+1}; that is, if a transition τ is enabled at all positions k of the computation $\bar{q}$, for $i \leq k \leq j$, and τ is not taken at any position k, for $i \leq k < j$, then τ is enabled continuously for time $t_{j+1} - t_i$. Therefore, the time sequence $\bar{t}$ is *consistent* with the computation $\bar{q}$ iff for every transition $\tau \in T$,

[*Lower bound*] if $taken(\tau) \in q_j$, then for all positions i with $0 \leq i < j$ and $t_{j+1} - t_i < \ell_\tau$, $enabled(\tau) \in q_i$ and $taken(\tau) \notin q_i$;

[*Upper bound*] if $enabled(\tau) \in q_k$ for all positions k with $i \leq k \leq j$, and $taken(\tau) \notin q_k$ for all positions k with $i \leq k < j$, then $t_{j+1} - t_i \leq u_\tau$.

A *timed computation* $(\bar{q}, \bar{t})$ of the timed transition system $P_{\ell,u}$ consists of a computation $\bar{q}$ of P together with a consistent time sequence $\bar{t}$. The first component of each timed computation of $P_{\ell,u}$ is an *untimed computation* of $P_{\ell,u}$. We write $\Pi_{\ell,u}(P)$ for the set of untimed computations of $P_{\ell,u}$. In general, $\Pi_{\ell,u}(P)$ is a strict subset of $\Pi(P)$; that is, the timing information ℓ and u plays the same role as fairness, namely, the role of restricting the admissible interleavings of enabled transitions. If Q is finite then, like $\Pi(P)$, the set $\Pi_{\ell,u}(P)$ is ω-regular [Alur and Dill 1994] (but not necessarily safe).

While the timed computations are required for checking if a system satisfies a specification that refers to time, the untimed computations suffice for checking if the system satisfies an untimed specification $\Phi \subseteq Q^\omega$: the timed transition system $P_{\ell,u}$ *satisfies* the specification Φ iff $\Pi_{\ell,u}(P) \subseteq \Phi$. In the unknown-delay model, we do not know the bound functions ℓ and u, but rather wish to prove that a transition system P satisfies the specification Φ for all possible choices of bound functions; that is, we wish to prove that the union $\bigcup_{\ell,u} \Pi_{\ell,u}(P)$ is contained in Φ. The following theorem shows that in order to verify a system in the unknown-delay model, it suffices to verify the system under finitary weak fairness; that is, the union $\bigcup_{\ell,u} \Pi_{\ell,u}(P)$ is the same as the set $\Pi_{fin(WF)}(P)$.

THEOREM 7. *Let P be a transition system with transition set T, let WF be the weak-fairness requirement for P, and let $\overline{q}$ be a computation of P. Then $\overline{q} \in \Pi_{fin(WF)}(P)$ iff there are two functions ℓ and u from T to $\mathbb{Q}_{>0}$ such that $\overline{q} \in \Pi_{\ell,u}(P)$.*

PROOF. Consider a weakly bounded computation $\overline{q} \in \Pi_{fin(WF)}(P)$. From Theorem 6, there is a positive integer k such that the schedule corresponding to $\overline{q}$ is weakly k-bounded. Let the bound functions be defined as $\ell_\tau = 1$ and $u_\tau = k+1$ for every transition τ. Consider the time sequence $\overline{t} = (0, 1, 2, 3, \ldots)$ (i.e., $t_i = i$ for all $i \geq 0$). Since time increases by 1 at every position, it is clear that the lower bound requirement is trivially satisfied. Since $\overline{q}$ is weakly k-bounded, no transition is enabled for more than k consecutive positions (and hence, for more than $k+1$ time units) without being taken. Thus, the consistency requirements are satisfied, and $(\overline{q}, \overline{t})$ is a timed computation of $P_{\ell,u}$, implying $\overline{q} \in \Pi_{\ell,u}(P)$.

To prove the converse, suppose that $\overline{q} \in \Pi_{\ell,u}(P)$ for some choice of ℓ and u. Let $\overline{t}$ be a time sequence such that $(\overline{q}, \overline{t})$ is a timed computation of $P_{\ell,u}$. Let the number of transitions in T be n. Let ℓ^* be the (nonzero) minimum of all the lower bounds ℓ_τ, and u^* be the (finite) maximum of all the upper bounds u_τ. Let k be an integer such that $k > n \cdot u^*/\ell^*$. We claim that the schedule corresponding to $\overline{q}$ is weakly k-bounded. Suppose not. Then there is a transition τ and a position $i \geq 0$ such that τ is enabled but not taken at all positions j of $\overline{q}$ with $i \leq j \leq i+k$. At every position of $\overline{q}$, $taken(\tau')$ holds for some transition τ'. Since $k > n \cdot u^*/\ell^*$, it follows that there is a transition τ' such that $taken(\tau')$ holds at more than u^*/ℓ^* distinct positions j of $\overline{q}$ with $i \leq j \leq i+k$. Since $\ell(\tau') \geq \ell^*$, from the assumption that $\overline{t}$ satisfies the lower-bound requirement of consistency, we have $t_{i+k} - t_i > u^*$, and therefore $t_{i+k} - t_i > u(\tau)$. But this implies that $\overline{t}$ violates the upper-bound requirement of consistency, a contradiction. In conclusion, $\overline{q}$ is weakly k-bounded, and hence, $\overline{q} \in \Pi_{fin(WF)}(P)$. □

We point out that all lower bounds are, although arbitrarily small, nonzero, and all upper bounds are finite. This is necessary, and justified because we universally quantify over all choices of bound functions. We also point out that reasoning in a timing-based model with specific bound functions—i.e., reasoning about timed computations—can be significantly more complicated than untimed reasoning [Henzinger et al. 1994]. Our analysis shows, therefore, that the verification of specifications that do not refer to time is conceptually simpler in the unknown-delay model than in the known-delay model.

3.5 The gap between finitary and standard fairness

The definition of finitary fairness replaces a given ω-language Π by the union of all ω-regular safety properties contained in Π. While this definition seems satisfactory in practice, there are obvious mathematical generalizations. We only point to some of the possibilities, without developing them in detail.

First, observe that the (uncountable) union of all safety properties contained in Π is Π itself. Not all safety properties, however, are definable by programs. We can obtain the *computable restriction* $com(\Pi)$ of Π by taking the (countable) union of all recursively enumerable (r.e.) safety properties that are contained in Π (an ω-language is *r.e.* iff it is generated by a nonhalting Turing machine). Clearly, $com(\Pi)$ captures all possible implementations of Π, finite-state or not, and typically falls strictly between $fin(\Pi)$ and Π. Computable fairness, however, does not have the two advantages of finitary fairness, namely, simpler verification rules and a solvable consensus problem.

There are also alternatives between $fin(\Pi)$ and $com(\Pi)$, which capture all implementations of Π with limited computing power. Recall the sample program P_0 from Section 2. For every schedule in $fin(\Pi)$, there is a bound, unknown but fixed, on how long a transition can be postponed. Suppose that we let this bound vary, and call a schedule *linearly bounded* iff the bound is allowed to increase linearly with time. While every bounded schedule is linearly bounded, the schedule $lrllrlllrllllr\ldots$ is linearly bounded but not bounded. In general, given a function $f(n)$ over the natural numbers, a schedule $\overline{w} \in (l+r)^\omega$ is $O(f)$-*bounded* iff there exists a constant k such that each of the two transitions l and r appears at least once in the subsequence $w_i w_{i+1} \ldots w_{i+k \cdot f(i)}$, for all $i \geq 0$. Finitary fairness, then, is $O(1)$-fairness. Moreover, for any fairness requirement F, we obtain a strict hierarchy of stronger fairness requirements $f(F)$, where $f(F)$ is the union of all $O(f)$-bounded schedulers that are contained in F. The algorithm presented in Section 5 can be modified so that it solves distributed consensus under the fairness requirement $f(F)$ for any fixed, computable choice of f.

4. APPLICATION: PROGRAM VERIFICATION

We now consider the problem of verifying that a program satisfies a specification under a finitary fairness assumption.

4.1 Model checking

If all program variables range over finite domains, then the set of program states is finite. The problem of verifying that such a finite-state program satisfies a temporal-logic specification is called *model checking* [Clarke and Emerson 1981]. Automated tools for model checking have been used successfully to check the correctness of digital hardware and communication protocols [Clarke and Kurshan 1996]. Here we examine the effects of finitary fairness on the algorithms that underlie these tools.

Untimed systems

Consider a transition system P with finite state set Q. The set $\Pi(P) \subseteq Q^\omega$ of computations of P is an ω-regular safety property. Since Q is finite, we can choose the scheduling alphabet to be Q itself. Let $F \subseteq Q^\omega$ be an ω-regular fairness requirement, and let $\Phi \subseteq Q^\omega$ be an ω-regular specification (given, say, by a PTL formula or a Büchi automaton). The verification question, then, is a problem of language inclusion: P satisfies Φ under F iff $\Pi(P) \cap F \subseteq \Phi$. This problem can be solved algorithmically, because all involved languages are ω-regular. Assuming finitary fairness, we need to check the language inclusion $\Pi(P) \cap \mathit{fin}(F) \subseteq \Phi$. It is, however, not obvious how to check this, because $\mathit{fin}(F)$ is not necessarily ω-regular. The following theorem shows that finite-state verification under finitary fairness can be reduced to verification under standard fairness.

THEOREM 8. *For all ω-regular languages Π_1 and Π_2, the set $\Pi_1 \cap \Pi_2$ is empty iff $\mathit{fin}(\Pi_1) \cap \Pi_2$ is empty.*

PROOF. We have $\mathit{fin}(\Pi_1) \cap \Pi_2 \subseteq \Pi_1 \cap \Pi_2$. Hence, if $\Pi_1 \cap \Pi_2$ is empty, then so is $\mathit{fin}(\Pi_1) \cap \Pi_2$. Now suppose that $\Pi_1 \cap \Pi_2$ is nonempty. Since both Π_1 and Π_2 are ω-regular, so is $\Pi_1 \cap \Pi_2$, and hence, it contains a word $\overline{w}$ such that $\overline{w} = \overline{w}_1 \overline{w}_2^\omega$ for (finite) strings $\overline{w}_1$ and $\overline{w}_2$. The language containing the single word $\overline{w}$ is safe, ω-regular, and contained in Π_1. Hence $\overline{w} \in \mathit{fin}(\Pi_1)$, and therefore $\overline{w} \in \mathit{fin}(\Pi_1) \cap \Pi_2$, implying that $\mathit{fin}(\Pi_1) \cap \Pi_2$ is nonempty. □

As a corollary we obtain that for model checking under finitary fairness, we can continue to use the algorithms that have been developed to deal with standard fairness.

COROLLARY 9. *Let P be a transition system with finite state set Q, and let $F, \Phi \subseteq Q^\omega$ be two ω-regular languages. Then P satisfies the specification Φ under the fairness requirement $\mathit{fin}(F)$ iff P satisfies Φ under the fairness requirement F.*

PROOF. We want to show that $\Pi(P) \cap \mathit{fin}(F) \subseteq \Phi$ iff $\Pi(P) \cap F \subseteq \Phi$. Let G be the ω-language $\Pi(P) \cap (Q^\omega \setminus \Phi)$. From the assumption that P is finite-state, $\Pi(P)$ is ω-regular. Since Φ is also ω-regular, so is G. Now $\Pi(P) \cap F \subseteq \Phi$ iff $F \cap G$ is empty iff $\mathit{fin}(F) \cap G$ is empty (by Theorem 8) iff $\Pi(P) \cap \mathit{fin}(F) \subseteq \Phi$. □

Timed systems

Consider a timed transition system $P_{\ell,u}$ with finite state set Q. Suppose that the specification does not refer to time, and is given as an ω-regular language $\Phi \subseteq Q^\omega$. To verify that $P_{\ell,u}$ satisfies Φ, we need to check the inclusion $\Pi_{\ell,u}(P) \subseteq \Phi$. This problem can be solved by constructing a Büchi automaton that recognizes the ω-regular language $\Pi_{\ell,u}(P)$ [Alur and Dill 1994]. The method is applicable, however, only when the bound functions ℓ and u are fully specified. If the bound maps are not fully specified, we obtain a parametric timing-verification problem [Alur et al. 1993]. The bound functions $\{\ell_\tau, u_\tau \mid \tau \in T\}$ are viewed as parameters: values of these parameters are not known, but are required to satisfy certain constraints (such as $\ell_{\tau_1} = u_{\tau_1}$ or $u_{\tau_2} < \ell_{\tau_3}$). The *parametric timing-verification problem*, then, is specified by

(1) a finite-state transition system $P = (Q, T, Q_0)$,
(2) an ω-regular specification $\Phi \subseteq Q^\omega$, and
(3) a set LU consisting of pairs (ℓ, u) of functions from T to $\mathbb{Q}_{>0}$.

The verification problem is to check that, for every choice of $(\ell, u) \in LU$, the resulting timed transition system $P_{\ell,u}$ satisfies the specification Φ. Define

$$\Pi_{LU}(P) = \bigcup_{(\ell,u)\in LU} \Pi_{\ell,u}(P).$$

Then, we want to check whether $\Pi_{LU}(P) \subseteq \Phi$. Theorem 8, together with Theorem 7, implies that the parametric timing-verification problem is decidable when the set LU consists of all function pairs.

COROLLARY 10. *Let P be a transition system with finite state set Q and transition set T, let $\Phi \subseteq Q^\omega$ be an ω-regular language, and let $LU = [T \mapsto \mathbb{Q}_{>0}]^2$ be the set of all pairs of functions from T to $\mathbb{Q}_{>0}$. The parametric timing-verification problem of checking the inclusion $\Pi_{LU}(P) \subseteq \Phi$ is decidable.*

This is in contrast to the general parametric timing-verification problem, which is undecidable if the set LU constrains, in simple ways, the legal choices of the bound functions [Alur et al. 1993]. For instance, if LU requires that $\ell_\tau = u_\tau$ for each transition $\tau \in T$ (i.e., $LU = \{(\ell, \ell) \mid \ell \in [T \mapsto \mathbb{Q}_{>0}]\}$), then the parametric timing-verification problem is undecidable.

4.2 Proof rules for termination

We now turn to the verification of programs that are not finite-state. Since safety specifications are proved independent of any fairness assumptions, we need to be concerned only with liveness specifications. We limit ourselves to proving the termination of programs (or, equivalently, to proving specifications of the form $\Diamond p$ for a state predicate p) under finitary weak fairness. It is straightforward to extend the proposed method to the verification of arbitrary temporal-logic specifications under the finitary versions of both weak and strong fairness.

Total termination versus just termination

Let $P = (Q, T, Q_0)$ be a transition system. The standard method for proving the termination of sequential deterministic programs can be adopted to prove that all computations of the (nondeterministic) transition system P terminate, which is called the *total termination* of P. Essentially, we need to identify a well-founded domain $(W, \prec)$ and a ranking (variant) function from the program states to W such that the rank decreases with every program transition. As an example, consider the rule **T** from [Lehman et al. 1982].

> Rule **T** for proving total termination:
>
> Find a ranking function ρ from Q to a well-founded domain $(W, \prec)$, and a state predicate $R \subseteq Q$. Then, show:
>
> (T1) $Q_0 \subseteq R$.
>
> (T2) For all states $q, q' \in Q$ and all transitions $\tau \in T$, if $R(q)$ and $q' \in \tau(q)$, then $R(q')$ and $\rho(q') \prec \rho(q)$.

The rule **T** is complete for proving total termination; that is, all computations of a transition system P terminate iff the rule **T** is applicable [Lehman et al. 1982]. Furthermore, it is always sufficient to choose the set $\mathbb{N}$ of natural numbers as the well-founded domain W.

Now consider the requirement that all weakly fair computations of P terminate, which is called the *just termination* of P. While the rule **T** is obviously sound for proving just termination, it is not complete. The problem is that there may not be a ranking function that decreases with every program transition. The standard solution is to identify a ranking function that never increases, and that is guaranteed

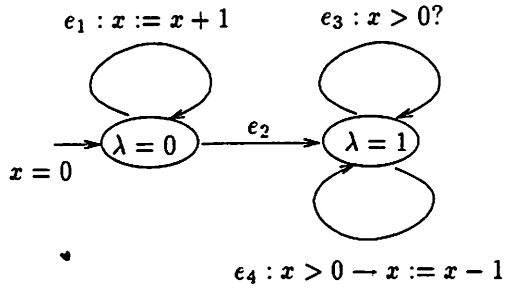

Fig. 1. Program P_1

to decrease eventually. The decrease is caused by so-called "helpful" transitions, whose occurrence is ensured by the weak-fairness requirement. As an example, consider the rule **J** from [Lehman et al. 1982].

> Rule J for proving just termination:
> Find a ranking function ρ from Q to a well-founded domain $(W, \prec)$, and a set R_τ of state predicates, one for each transition $\tau \in T$. Let R be the union of all R_τ for $\tau \in T$. Show for all states $q, q' \in Q$ and all transitions $\tau, \tau' \in T$:
> (J1) $Q_0 \subseteq R$.
> (J2) If $q \in R$ and $q' \in \tau(q)$, then $q' \in R$ and $\rho(q') \preceq \rho(q)$.
> (J3) If $q \in R_\tau$ and $q' \in \tau(q)$, then $\rho(q') \prec \rho(q)$.
> (J4) If $q \in R_\tau$ and $q' \in \tau'(q)$ and $\rho(q) = \rho(q')$, then $q' \in R_\tau$.
> (J5) If $q \in R_\tau$ and some transition is enabled in q, then τ is enabled in q.

The rule **J** is complete for proving just termination: all weakly fair computations of a transition system P terminate iff the rule **J** is applicable. Completeness, however, no longer holds if we require the well-founded domain to be the set $\mathbb{N}$ of natural numbers: there are transition systems for which transfinite induction over ordinals higher than ω is needed to prove just termination.

An example

Before we present the method for proving termination of all finitarily fair computations, let us consider an example. Consider the transition system P_1 of Figure 1. A state of the program P_1 is given by the values of its two variables: the location variable λ ranges over $\{0, 1\}$, and the data variable x is a nonnegative integer; initially $\lambda = x = 0$. The four transitions e_1, e_2, e_3, e_4 are as shown in the figure.

We want to prove that all weakly fair computations of P_1 terminate. Initially $\lambda = 0$, and both transitions e_1 and e_2 are continuously enabled. Fairness to e_2 ensures that eventually $\lambda = 1$. Since e_4 is enabled as long as x is positive, and decrements x each time, fairness to e_4 ensures that eventually $x = 0$, resulting in termination. To prove termination formally, we apply the rule **J**. As the well-founded domain, we choose the set $\mathbb{N} \cup \{\omega\}$ of natural numbers together with the ordinal ω. Choose both R_{e_1} and R_{e_3} to be the empty set; a state (λ, x) belongs

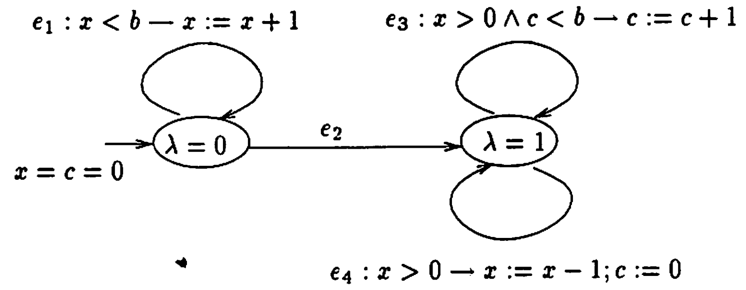
$e_1 : x < b \rightarrow x := x + 1$
$e_3 : x > 0 \wedge c < b \rightarrow c := c + 1$
$\lambda = 0$
e_2
$\lambda = 1$
$x = c = 0$
$e_4 : x > 0 \rightarrow x := x - 1; c := 0$

Fig. 2. Transformed program *fin*(P_1)

to R_{e_2} iff $\lambda = 0$, and belongs to R_{e_4} iff $\lambda = 1$. The ranking function is defined as: $\rho(0, x) = \omega$, and $\rho(1, x) = x$. The transitions e_1 and e_3 leave the rank unchanged, while e_2 and e_4 cause a decrease. The reader can check that the five premises (J1)–(J5) of the rule **J** are indeed satisfied. Notice that there is no bound on the number of steps, as a function of the initial values of the variables, before P_1 terminates. This *unbounded termination* is what makes the mathematical treatment of fairness difficult.

Proving the termination of all finitary weakly fair computations of P_1—that is, proving the *finitary just termination* of P_1—is conceptually simpler. Recall that for every computation in *fin*(*WF*), there is an integer k such that a transition cannot be enabled continuously for more than k steps without being taken (Theorem 6). It follows that, under finitary weak fairness, P_1 must terminate within a bounded number of steps, where the bound depends on the unknown constant k. To capture this intuition, we transform the program P_1 by introducing the two auxiliary variables b and c. The initial value of the variable b is an unspecified positive integer, and the program transitions do not change its value. The integer variable c is used to ensure that no transition is enabled for more than b steps without being taken. We thus obtain the new program *fin*(P_1) of Figure 2.

The original program P_1 terminates under the finitary weak-fairness requirement *fin*(*WF*) iff all computations of the transformed program *fin*(P_1) terminate. Thus, we have reduced the problem of proving the finitary just termination of P_1 to the problem of proving the total termination of *fin*(P_1). Hence, the simple rule **T** with induction over the natural numbers is sufficient to prove finitary just termination. A state of *fin*(P_1) is a tuple (λ, x, c, b). To apply the rule **T**, we choose the set R to be the set of reachable states: $(0, x, c, b) \in R$ iff $c = 0$ and $x \leq b$; and $(1, x, c, b) \in R$ iff $c \leq b$ and $x \leq b$. The ranking function is a mapping from R to the natural numbers defined by $\rho(0, x, 0, b) = 2b^2 + 2b + 1 - x$, and $\rho(1, x, c, b) = 2bx + b - c$. The reader should check that every transition, applied to any state in R, causes the ranking function to decrease.

Notice that the transformed program *fin*(P_1) has infinitely many initial states, but for any given initial state, it terminates within a bounded number of steps. Consequently, *fin*(P_1) does not suffer from the problems caused by unbounded termination. The tradeoff between proving just termination of P_1 and total termination of *fin*(P_1) should be clear: while the rule **J** used for the former is more complex than the rule **T** used for the latter, the program $fin(P_1)$ is more complex than P_1. This is similar in spirit to proving fair termination by introducing explicit schedulers [Apt et al. 1984].

The finitary transformation of a program

We now present a generic transformation for proving the finitary just termination of a transition system $P = (Q, T, Q_0)$ using the rule **T**. Let T consist of m transitions $\tau_1, \ldots, \tau_m$. The *finitary transformation* $fin(P)$ of the transition system P is obtained by introducing a new positive integer variable b (the "bound"), and for each transition τ_j, where $1 \leq j \leq m$, a new nonnegative integer variable c_j (the "counters"). Thus, the state space of $fin(P)$ is $Q \times \mathbb{N}_{>0} \times \mathbb{N}^m$. The initial value of the bound b is arbitrary; the initial value of each counter c_j is 0. Thus, the set of initial states of $fin(P)$ is $Q_0 \times \mathbb{N}_{>0} \times \{0\}^m$. For every transition $\tau \in T$, the transition system $fin(P)$ contains a transition $fin(\tau)$ such that $(q', b', c'_1, \ldots, c'_m) \in fin(\tau)(q, b, c_1, \ldots, c_m)$ iff

(1) $q' \in \tau(q)$,
(2) $b' = b$, and
(3) for all $1 \leq j \leq m$, if $\tau_j(q)$ is empty or $q' \in \tau_j(q)$ then $c'_j = 0$, else $c_j < b$ and $c'_j = c_j + 1$.

The following theorem establishes the transformation fin together with the simple rule **T** as a sound and complete proof method for finitary just termination.

THEOREM 11. *The finitary weakly fair computations of the transition system P terminate iff all computations of the transition system $fin(P)$ terminate.*

PROOF. Suppose that the program $fin(P)$ has a nonterminating computation $\overline{q}$. Consider the projection $\overline{q}'$ of $\overline{q}$ on the state space of P. From the transition rules of $fin(P)$, it is clear that $\overline{q}'$ is a computation of P. The value of the bound variable b stays unchanged throughout $\overline{q}$: let it be k. Furthermore, $c_j \leq k$ is an invariant over the computation $\overline{q}$ for all $1 \leq j \leq m$. Since for each transition τ_j, the counter variable c_j is incremented each time τ_j is enabled but not taken, it follows that the computation $\overline{q}'$ is weakly k-bounded. Hence, P has a weakly fair nonterminating computation.

Conversely, consider a weakly fair nonterminating computation $\overline{q}$ of P. From Theorem 6, the computation $\overline{q}$ is weakly k-bounded for some k. Define the sequence $\overline{q}'$ over the state space of $fin(P)$ as follows. For all positions $i \geq 0$, let $q'_i = (q_i, k, c^i_1, \ldots, c^i_m)$ be such that c^i_j is the maximal nonnegative integer $n \leq i$ such that the transition τ_j is enabled but not taken at all positions n' with $i - n \leq n' < i$. It is easy to check that, since $\overline{q}$ is weakly k-bounded, if the transition τ_j is enabled but not taken at position i, then $c^i_j < k$. Consequently, $\overline{q}'$ is a (nonterminating) computation of $fin(P)$. □

Thus, the language $\Pi_{fin(WF)}(P)$ of the finitary weakly fair computations of the transition system P is the projection of the language $\Pi(fin(P))$ of the transformed transition system $fin(P)$. It is known that given a transition system P and a fairness requirement F, there exists a transition system P' such that $\Pi_F(P) = \Pi(P')$ [Vardi 1987]. This transformation, however, requires uncountably many states (the transformed system P' has one initial state for every fair computation in F), and does not yield a proof principle for which well-founded induction over $\mathbb{N}$ is adequate.

5. APPLICATION: DISTRIBUTED CONSENSUS

We consider the consensus problem in a shared-memory model where the only atomic operations allowed on a shared register are read and write. Formally, the *consensus problem* is defined as follows. There are n processes P_i, for $1 \leq i \leq n$, each with a boolean input value $in_i \in \{0, 1\}$. The process P_i decides on the value $v \in \{0, 1\}$ by executing the statement *decide*(v). To model failures, we introduce a special transition $fail_i$ for each process. The transition $fail_i$ is enabled only if the process P_i has not yet decided on a value. When P_i takes the transition $fail_i$, all of its transitions are disabled, and P_i stops participating.

A solution to the consensus problem must satisfy *agreement*—that is, no two processes decide on conflicting values—and *validity*—that is, if a process decides on the value v, then v is equal to the input value of some process. Apart from these two safety properties, we want the nonfailing processes to decide eventually: *wait-freedom* asserts that each process P_i eventually either decides on some value or fails. Thus a process must not prevent another process from reaching a decision, and the algorithm must tolerate any number of process failures. The implicit fairness assumption in the asynchronous model is the weak-fairness requirement WF for all program transitions except the newly introduced $fail_i$ transitions. It is known that, even for $n = 2$, there is no program that satisfies all three consensus properties under the weak-fairness requirement WF [Fischer et al. 1985; Loui and Abu-Amara 1987].

On the other hand, consensus can be solved in the unknown-delay model, where it is assumed that there is an upper bound Δ on memory-access time, but the bound is unknown to the processes a priori and a solution is required to work for all values of Δ [Alur et al. 1997]. We show that the consensus algorithm of [Alur et al. 1997] for the unknown-delay model solves, in fact, consensus under the finitary weak-fairness requirement $fin(WF)$. The algorithm is shown in Figure 3. The algorithm proceeds in rounds and uses the following shared data structures: an infinite two-dimensional array $x[*, 2]$ of bits, and an infinite array $y[*]$ whose elements have the value $\bot$, 0, or 1. The decision value (i.e., the value that the processes decide on) is written to the shared bit *out*, which initially has the value $\bot$. In addition, each process P_i has a local register v_i that contains its current preference for the decision value, and a local register r_i that contains its current round number.

If all processes in a round r have the same preference v, then the bit $x[r, \neg v]$ is never set to 1, and consequently, processes decide on the value v in round r. Furthermore, if a process decides on a value v in round r, then $y[r]$ is never set to the conflicting value $\neg v$, and every process that reaches round $r + 1$ has the preference v for that round. This ensures the agreement property (see [Alur et al. 1997] for more details of the proof). It is easy to check that if all processes have the same initial input v, then no process will ever decide on $\neg v$, implying the validity property.

It is possible that two processes with conflicting preferences for round r cannot resolve their conflict in round r, and proceed to round $r + 1$ also with conflicting preferences. This happens only if both of them find $y[r] = \bot$ first (line 3), and one of

```
Shared registers: initially $out = \bot$, $y[1..] = \bot$, $x[1..,0..1] = 0$;
Local registers: initially $r_i = 1$, $v_i = in_i$;
1. while $out = \bot$ do
2.        $x[r_i, v_i] := 1$;
3.        if $y[r_i] = \bot$ then $y[r_i] := v_i$ fi;
4.        if $x[r_i, \neg v_i] = 0$ then $out := v_i$
5.                        else  for $j = 1$ to $r_i$ do skip od;
6.                              $v_i := y[r_i]$;
7.                              $r_i := r_i + 1$
8.                        fi
9.        od;
10. $decide(out)$.
```

Fig. 3. Consensus. assuming finitary weak fairness (program for process P_i with input in_i)

them proceeds and chooses its preference for the next round (line 7) before the other one finishes the assignment to $y[r]$. The finitary weak-fairness requirement ensures that this behavior cannot be repeated in every round. In every finitary weakly-fair computation, there is a bound k such that every process that has neither failed nor terminated takes a step at least once every k steps. Once the round number exceeds the (unknown) bound k, while a process is executing its **for** loop, all other processes are forced to take at least one step. This suffices to ensure termination.

THEOREM 12. *The program of Figure 3 satisfies the properties of agreement, validity, and wait-freedom under the finitary weak-fairness requirement fin(WF).*

By contrast, the program does not satisfy wait-freedom under the standard weak-fairness requirement *WF*. Also observe that the algorithm uses potentially unbounded space, and therefore is not a finite-state program. Corollary 9, together with the impossibility result for consensus in the asynchronous shared-memory model, implies that there is no algorithm that uses a fixed number of bounded registers and solves consensus under finitary fairness.

THEOREM 13. *For two processes. there is no algorithm that uses finite memory and satisfies the properties of agreement, validity, and wait-freedom under the finitary weak-fairness requirement fin(WF) (or equivalently. in the unknown-delay model).*

The unknown-delay model of [Dwork et al. 1988] consists of distributed processes communicating via messages. where the delivery time for each message is bounded, but is not known a priori. They establish bounds on the number of process failures that can be tolerated by consensus protocols under various fault models. By Theorem 7, these bounds can also be established using finitary weak fairness. A similar observation applies to the results on the *session problem* for the unknown-delay model [Rhee and Welch 1992].

6. CONCLUSIONS

To establish liveness properties of concurrent programs. one needs fairness assumptions about the scheduling of enabled transitions. We have proposed to revisit the standard definitions of fairness and consider, instead, the corresponding finitary versions. Intuitively, where standard fairness means "infinitely often," finitary fairness means "at least once every k steps, for an unknown constant k." The main contribution of the paper is the mathematical formulation of this intuitive concept. The finitary restriction $fin(F)$ of any given ω-language F is the union of all ω-regular safety properties contained in F. This definition allows us to define finitary versions of different fairness requirements such as weak and strong fairness.

To illustrate that the proposed notion is adequately abstract, we have shown that finitary fairness is equivalent to, in a precise mathematical sense, timing-based models with unknown bounds on delays. Two advantages of finitary fairness over the standard definitions have been demonstrated. First, we have given a program transformation that reduces proving termination under finitary fairness to total termination. This implies that, unlike termination under standard fairness, termination under finitary fairness can always be established using well-founded induction over the natural numbers. Second, strengthening the assumption from standard fairness to finitary fairness allows the design of algorithms for problems that are unsolvable under standard fairness. This is illustrated by the fact that the fundamental problem of consensus in a faulty asynchronous distributed environment can be solved assuming finitary fairness.

Acknowledgments

Notions that are similar to k-bounded fairness, for a fixed k, have been defined in several places [Jayasimha 1988]; the notion of bounded fairness seems to be part of the folklore, but we do not know of any published account. We thank Leslie Lamport, Amir Pnueli, Fred Schneider, Gadi Taubenfeld, and Sam Toueg for pointers to the literature and for helpful discussions. We also thank the anonymous reviewers for valuable suggestions.

REFERENCES

Alpern, B., Demers, A., and Schneider, F. 1986. Safety without stuttering. *Information Processing Letters 23*, 177–180.

Alpern, B. and Schneider, F. 1985. Defining liveness. *Information Processing Letters 21*, 181–185.

Alur, R., Attiya, H., and Taubenfeld, G. 1997. Time-adaptive algorithms for synchronization. *SIAM Journal on Computing 26*, 539–556.

Alur, R. and Dill, D. 1994. A theory of timed automata. *Theoretical Computer Science 126*, 183–235.

Alur, R., Henzinger, T., and Vardi, M. 1993. Parametric real-time reasoning. In *Proceedings of the 25th ACM Symposium on Theory of Computing*. 592–601.

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

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

Büchi, J. 1962. On a decision method in restricted second-order arithmetic. In *Proceedings of the International Congress on Logic, Methodology, and Philosophy of Science 1960*. Stanford University Press. 1–12.

Clarke, E. and Emerson, E. 1981. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Proceedings of the Workshop on Logic of Programs*. Lecture Notes in Computer Science 131. Springer-Verlag. 52–71.

Clarke, E. and Kurshan, R. 1996. Computer-aided verification. *IEEE Spectrum 33*, 61–67.

Clarke, E. and Wing, J. 1996. Formal methods: State of the art and future directions. *ACM Computing Surveys 28*, 626–643.

Dwork, C., Lynch, N., and Stockmeyer, L. 1988. Consensus in the presence of partial synchrony. *Journal of the ACM 35*, 288–323.

Fischer, M., Lynch, N., and Paterson, M. 1985. Impossibility of distributed consensus with one faulty process. *Journal of the ACM 32*, 374–382.

Francez, N. 1986. *Fairness.* Springer-Verlag.

Gabbay, D., Pnueli, A., Shelah, S., and Stavi, J. 1980. On the temporal analysis of fairness. In *Proceedings of the Seventh ACM Symposium on Principles of Programming Languages.* 163–173.

Henzinger, T., Manna, Z., and Pnueli, A. 1994. Temporal proof methodologies for timed transition systems. *Information and Computation 112*, 273–337.

Jayasimha, D. 1988. *Communication and Synchronization in Parallel Computation.* Ph.D. Thesis, University of Illinois at Urbana-Champaign.

Lehman, D., Pnueli, A., and Stavi, J. 1982. Impartiality, justice, and fairness: The ethics of concurrent termination. In *Proceedings of the Ninth International Colloquium on Automata, Languages, and Programming.* Lecture Notes in Computer Science 115. Springer-Verlag, 264–277.

Loui, M. and Abu-Amara, H. 1987. Memory requirements for agreement among unreliable asynchronous processes. *Advances in Computing Research 4*, 163–183.

Manna, Z. and Pnueli, A. 1990. A hierarchy of temporal properties. In *Proceedings of the Ninth ACM Symposium on Principles of Distributed Computing.* 377–408.

Manna, Z. and Pnueli, A. 1991. *The Temporal Logic of Reactive and Concurrent Systems: Specification.* Springer-Verlag.

Merritt, M., Modugno, F., and Tuttle, M. 1991. Time-constrained automata. In *Proceedings of the Workshop on Theories of Concurrency.* Lecture Notes in Computer Science 527. Springer-Verlag, 408–423.

Pease, M., Shostak, R., and Lamport, L. 1980. Reaching agreement in the presence of faults. *Journal of the ACM 27*, 228–234.

Rhee, I. and Welch, J. 1992. The impact of time on the session problem. In *Proceedings of the 11th ACM Symposium on Principles of Distributed Computing.* 191–201.

Thomas, W. 1990. Automata on infinite objects. In *Handbook of Theoretical Computer Science*, J. van Leeuwen, Ed. Vol. B. Elsevier Science Publishers, 133–191.

Vardi, M. 1987. Verification of concurrent programs: The automata-theoretic framework. In *Proceedings of the Second IEEE Symposium on Logic in Computer Science.* 167–176.